Title
Cryptographic Techniques for Privacy Preserving Identity

Author
Bethencourt, John

Publication Date
2011

Peer reviewed|Thesis/dissertation

Cryptographic Techniques for Privacy Preserving Identity

by

John Daniel Bethencourt

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Associate Professor Dawn Song, Chair
Assistant Professor Deirdre Mulligan
Professor David Wagner

Spring 2011

Abstract

Cryptographic Techniques for Privacy Preserving Identity

by

John Daniel Bethencourt

Doctor of Philosophy in Computer Science

University of California, Berkeley

Associate Professor Dawn Song, Chair

Currently, people have a limited range of choices in managing their identity online. They can use their real name or a long-term pseudonym, thereby lending context and credibility to information they publish but retaining no control over their privacy, or they can post anonymously, ensuring strong privacy but lending no additional credibility to their posts. In this work, we aim to develop a new type of online identity that allows users to publish information anonymously and unlinkably while simultaneously backing their posts with the credibility offered by a single, persistent identity. We show how these seemingly contradictory goals can be achieved through a series of new cryptographic techniques.

Our consideration of the utility of persistent identities focuses on their ability to develop reputation. In particular, many online forums include systems for recording feedback from a user's prior behavior and using it to filter spam and predict the quality of new content. However, the dependence of this reputation information on a user's history of activities seems to preclude any possibility of anonymity. We demonstrate that useful reputation can, in fact, coexist with strong privacy guarantees by developing a novel cryptographic primitive we call "signatures of reputation" which supports monotonic measures of reputation in a completely anonymous setting. In our system, users can express trust in others by voting for them, collect votes to build up their own reputation, and attach a proof of their reputation to any data they publish, all while maintaining the unlinkability of their actions.

Effective use of our scheme for signatures of reputation requires a means of selectively retrieving information while hiding one's search criteria. The sensitivity of search criteria is widely recognized and has previously been addressed through a series of cryptographic schemes for private information retrieval (PIR). Among the more recent of these is a scheme proposed by Ostrovsky and Skeith for a variant of PIR termed "private stream searching." In this setting, a client encrypts a set of search keywords and sends the resulting query to an untrusted server. The server uses the query on a stream of documents and returns those that match to the client while learning nothing about the keywords in the query. To retrieve documents of total length n, the Ostrovsky-Skeith scheme requires the server to return data

of length $O(n \log n)$. We present a new private stream searching scheme that improves on this result in several ways. First, we reduce the asymptotic communication to $O(n + m \log m)$, where $m \leq n$ is the number of distinct documents returned. More importantly, our scheme improves the multiplicative constants, resulting in an order of magnitude reduction in communication in typical scenarios. We also provide several extensions to our scheme which increase its flexibility and correspondingly broaden its applicability.

With the help of our private stream searching scheme, the proposed signatures of reputation allow users to accumulate positive feedback over time and attach a proof of their current reputation to any information they post online, all while maintaining the unlinkability of their actions. Because of the unlinkability provided, the user is free to use a single identity across all applications, thereby obtaining the most reputation. A detailed analysis of practical performance shows that our proposals, while costly, are within the capabilities of present computing and communications infrastructure.

We conclude our investigation into the potential for new forms of online identity with an evaluation of what might be considered the final frontier in attacks on anonymity: the possibility of linking posted information to its author solely through its content. Even if all explicit forms of identity are stripped from information a user posts online, it must remain intelligible to others to be useful. In the case of textual content, we note that the techniques of stylometry might allow an adversary to determine the likely author of an anonymous post by comparing it to material previously posted elsewhere. Through a series of large-scale experiments we show that, in some cases, this is indeed possible, and that individuals who have authored large amounts of content already online are the most vulnerable.

Contents

List of Figures

List of Tables

Acknowledgments

A number of individuals and institutions contributed directly and indirectly to the work presented here. First, I owe a great deal of gratitude to my advisor, Dawn Song. For six years, her ideas have provided me with fertile ground for research. In her role as my advisor, I couldn't have asked for more: always ready with guidance, yet willing to step aside when I wished to explore on my own. Brent Waters deserves similar recognition; he has effectively been a second advisor to me. Among my other collaborators, I would also like to single out Elaine Shi for my thanks, as her help was especially crucial to the success of this work.

I am grateful to David Wagner and Deirdre Mulligan for agreeing to serve on my committee with Dawn. If you have met either of them, you are no doubt aware of their unusually kind and helpful dispositions—I can only hope that others do not exploit their generosity excessively.

I am very fortunate to have had the privilege to study and conduct research at the University of California in Berkeley and at Carnegie Mellon University, where I began my graduate studies before moving with my advisor to Berkeley. I am also grateful for the support I have received from the National Science Foundation and the Department of Defense in the form of graduate fellowships.

Chapter 1

Privacy Preserving Identity

Our increasingly connected world offers an ever broadening array of ways to share information. Unlike the web users of ten years ago, today's users can make additions to most everything they can consume. In addition to the newer forms of communication offered by wikis and social networks, most conventional websites now include commenting facilities. While the increase in opportunities for public participation has obvious benefits, it also presents users with new challenges in managing their identity and corresponding threats to their privacy. Currently, users have three choices when attaching their identity to information they publish online: they can use their real name, a pseudonym, or nothing at all, posting anonymously.

A few applications require or encourage users to use their real name. For example, the policy of the online edition of the Wall Street Journal includes the following statement. "The Journal Community encourages thoughtful dialogue and meaningful connections between real people. We require the use of your full name to authenticate your identity. The quality of conversations can deteriorate when real identities are not provided." [85] Encouraging thoughtful dialogue is an important goal, but in many situations this policy may not be feasible. A message board intended as, say, a support group for victims of abuse would significantly limit participation if it required users to reveal their real name. It is easy to imagine many other scenarios in which a user has a strong, legitimate interest in hiding their identity.

Posting information completely anonymously, on the other hand, helps ensure the author's privacy but lacks the accountability provided by persistent identities. Such information can be compared to writing found on the wall of a public restroom: its credibility is limited to whatever might be discernible from the content itself.

Use of one or more pseudonyms (which may or may not be shared across applications) falls between these two extremes and is the most widely used alternative. If users are able create new pseudonyms at will, they have some flexibility in selectively linking themselves to their previous activities. However, the tradeoff between credibility and privacy remains. Using a large number of pseudonyms limits their utility, but a small number of pseudonyms

will be more easily linked to each other and to a user's true identity.

Much privacy research to date has focused on enabling posting with total anonymity, but the relative lack of applications used in this way suggests the importance of the benefits provided by persistent identities. Many of these benefits may be summarized with the observation that a persistent identity is capable of developing reputation. In addition to the implicit reputation which exists when a user simply remembers a previous encounter with a name or pseudonym, systems which explicitly manage various forms of reputation have become a ubiquitous tool for improving the quality of online interactions. For example, a user may mark a product or business review as "useful," and these ratings allow others to more easily identify the best reviews and reviewers. Most web message boards also include a means of providing feedback to help filter spam and highlight quality content. In some cases, reputation is crucial to the basic function of an application—most online auctions would not take place in the absence of a system for feedback between buyers and sellers.

1.1 Overview of Contributions

This work aims to develop a new type of online identity that combines the best features of all currently available options: the unlinkable nature of completely anonymous activity and the credibility offered by a single identity with a persistent reputation. That is, we wish to allow users to publish and retrieve information and develop a reputation that lends credibility to their posts, all without explicitly identifying themselves or linking their actions together.

Such a system would enable a number of intriguing applications. For example, we might imagine an anonymous message board in which every post stands alone—not even associated with a pseudonym. Users would rate posts based on whether they are helpful or accurate, collect reputation from other users' ratings, and annotate or sign new posts with the collected reputation. Other users could then judge new posts based on the author's reputation while remaining unable to identify the earlier posts from which it was derived. Such a forum would allow effective filtering of spam and highlighting of quality information while providing an unprecedented level of user privacy.

Our primary contributions are two sets of cryptographic tools which help enable such applications: one which assists in publishing information and another which assists in retrieving it. Additionally, we offer an investigation into certain threats to anonymity that may exist even if the proposed measures were to be used. We now give an overview of our contributions in each of these three areas of research.

Unlinkable reputation. Perhaps the most challenging of our goals is the development of a sort of anonymous reputation. It seems almost contradictory to require the ability to judge information by the reputation of its author while simultaneously hiding the author's identity and preventing the author's activities from being linked together. After all, a user's reputation must be derived from the history of that user's activities.

We enable this counter-intuitive situation through a new cryptographic framework we call "signatures of reputation." In a conventional digital signature scheme, a signature is associated with a public key and convinces the verifier that the signer knows the corresponding private key. Based on the public key, a verifier could then retrieve the reputation of the signer. Through signatures of reputation, we aim to eliminate the middle step of identifying the signer: instead, verification of the signature directly reveals the signer's reputation and convinces the verifier of its accuracy. With such a tool, a user can apply their reputation to any data that they wish to publish online, without risking their privacy.

By formally defining this setting, we hope to spur further research into techniques for its realization. As a first step, we have developed a construction for signatures of reputation that supports monotonic aggregation of reputation. That is, we assume that additional feedback cannot decrease a user's reputation. Although some existing reputation systems are monotonic (e.g., Google's PageRank algorithm [73] and many of the systems used by web message boards), one would ultimately hope to support non-monotonic reputation as well. However, support for non-monotonic reputation is a significantly more challenging problem which we consider to be beyond the scope of our present efforts.

In our construction, the reputation feedback takes the form of cryptographic "votes" that users construct and send to one another, and a user's reputation is simply the number of votes they have collected from distinct users. Each user stores the votes they have collected, and to anonymously sign a message with their reputation, the user constructs a non-interactive zero-knowledge (NIZK) proof of knowledge which demonstrates possession of some number of votes. The ability of a reputation system to limit the influence of any single user is crucial in enabling applications to control abuse. To this end, our construction ensures that each user can cast at most one valid vote for another user (or up to k for any fixed $k \geq 1$). Enforcing this property is a major technical problem due to the tension with the desired unlinkability properties; we solve it through a technique for proving the distinctness of a list of values within a NIZK. The size of the resulting signature is linear in the number of votes present. We also provide an alternative scheme which satisfies a weaker security requirement while using only logarithmic space.

Private searches. Complementing the ability to publish information anonymously and unlinkably is the ability to retrieve it under similar guarantees. In particular, searches based on one or more textual keywords have become the standard means by which users distill relevant information from the ever increasing volumes available. Like publishing content online, sending search criteria has privacy implications. However, many common queries can be more directly identifying. This is unsurprising when you consider the fact that users construct queries to retrieve information relevant to themselves, but they post content when they think it will be relevant to others. As an extreme example, it's a common practice to search the web for one's own name, which can immediately link your other queries to your identity (e.g., via a cookie or an IP address).

This issue poses a problem for our proposed signatures of reputation. While our scheme provides a means to prove possession of reputation feedback (as represented by cryptographic tokens), the user still needs a way to retrieve the feedback that has been applied to their posted content. Unless the user can do so without linking together their posts through their query, the properties provided by the signatures of reputation are lost.

Fortunately, the privacy of search criteria is well suited to cryptographic protection. While content intend for human consumption must remain comprehensible, the criteria provided to a search mechanism need only function correctly. If the search criteria can be encrypted while still somehow allowing the search to be carried out, the privacy of the user conducting the search can be ensured. Schemes for private information retrieval (PIR) provide exactly this capability. Traditionally, the problem of PIR is modeled as a bitstring (a "database" fixed ahead of time) which can be queried by specifying the index of a bit. The query is given to a server in an encrypted form and the server processes it with the bitstring to produce an encrypted result, which the client can decrypt to obtain the specified bit. A secure scheme for PIR ensures that the server is unable to determine which bit was retrieved.

Obviously, such a rudimentary search mechanism is too inflexible for most practical applications. Ostrovsky and Skeith have expanded this model to "private stream searching" [72]. The scheme they provide allows retrieval of fixed length documents or files rather than bits, and they may be selected using a disjunction of search keywords selected from a predetermined dictionary. Most significantly, the set of documents need not be fixed ahead of time: an encrypted query can be applied to any number of documents as they arise, and the results can be returned at any time. While much more flexible than standard PIR, their scheme is inefficient. In practice, it may inflate the size of data returned by factor of about fifty.

We have developed a new scheme for private stream searching that improves on this result in a number of ways. It is dramatically more efficient, incurring only a small overhead in the size of the returned data (typically a factor of about three), supports retrieval of variable length documents, and allows for distributed evaluation of searches. We also show how to search for arbitrary strings, avoiding the need for a predetermined dictionary, although this technique introduces the possibility of false positive matches. Given these improvements in efficiency and flexibility, our scheme might be considered the first method for private searching with a reasonable likelihood of being useful in a real application. In particular, because it allows searching for arbitrary strings, it is the only scheme which provides the means of privately retrieving reputation feedback that is necessary to use our signatures of reputation.

A further threat to anonymity. Would these two sets of techniques be sufficient to enable truly anonymous online communities? One threat comes to mind: the possibility of linking posted information to its author based solely on its content. Even if all explicit forms of identity are stripped from information a user posts online, it must remain intelligible to others to be useful. Can the human-readable information inherent in any communication be

exploited to identify its author? Our third area of research provides some initial answers to this question. Specifically, we have investigated whether the techniques of stylometry can be applied at a sufficiently large scale to threaten authors' privacy.

Stylometry is the study of author-correlated features of prose for the purpose of authorship attribution. Almost any features can be useful, from sentence length distributions, to letter n-gram frequencies, to idiosyncrasies such as spelling errors. The goal may be to match a text from an unknown author against a set of labeled examples (classification) or to group multiple unlabeled texts by author (clustering). These methods have long been employed in a literary context, when a text with unknown or disputed authorship is compared with texts from a small number of possible authors. However, to date, little investigation has been given to the possibility of stylometry threatening the privacy of modern Internet users.

To evaluate this threat, we assembled datasets of blog posts and simulated the task of matching an anonymous or pseudo-anonymous blog with others (if any) from the same author. The results of these experiments demonstrate that current machine learning techniques are perhaps surprisingly effective for large-scale authorship attribution. We also provide some insights into the relationship between the identifiability of an author and the amounts of labeled and unlabeled content that were posted.

1.2 Related Work and Existing Techniques

Before further describing the new techniques we have developed, we will now discuss several areas of related work. We first survey work that helps motivate our goals. Then, for each of the two sets of cryptographic tools described in the previous section, we discuss existing techniques that are similar to our approach at a technical level.

Motivation. The difficulties in managing one's identity as our lives become more connected are widely recognized. Using your real name wherever identity is needed poses obvious privacy risks, but there is also a growing understanding that pseudonyms provide inadequate protection. Recent work has shown that a small amount of prior information is often sufficient to match an individual to their pseudonym, for example, as in the case of the Netflix Prize movie rental dataset [71]. It was shown that knowledge of only a couple approximate movie rental dates (as might be revealed by simply mentioning what one has watched recently) is typically enough to uniquely identify an individual in the dataset, revealing their entire rental history. Work in a similar vein has shown that individuals can easily be matched to their identifier or pseudonym in anonymized social graphs [7]. The key failing highlighted by these examples is the fact that pseudonyms link together a user's activities and associations into a single history, thereby rapidly narrowing down the possibilities for their identity. When enough information is linked together, seemingly innocuous details become personally identifiable.

The sensitivity of search terms in particular has also attracted attention. In 2005 the U.S. Department of Justice caused controversy when it subpoenaed records of queries from popular web search engines. The especially identifying nature of search terms was highlighted again when AOL Research released a database of about 20 million search queries on the Internet, revealing a great deal of compromising information about 658,000 AOL users, who were often easy to identify based only on the content of their searches [3]. Note that other privacy preserving technologies such as mix-based anonymizers [40] do not solve this problem, since the search terms alone were enough to compromise the user's privacy. In general, previous research on providing anonymity has focused on lower-level networking, while our research concerns higher-level applications and content. In this respect, the two areas of work are complementary.

Signatures of reputation. While we are not aware of any work directly comparable to our proposed signatures of reputation, others have explored the conflict between reputation and unlinkability [82, 76, 81]. E-cash schemes [4, 29, 5, 26] also attempt to maintain the unlinkability of individual user interactions, and in several cases [9, 27, 2] they have been applied for reputation or incentive purposes. The work of Androulaki et al. [2] is particularly close to ours in its aims. However, this and all other e-cash based approaches are incapable of supporting the type of abuse resistance provided by our scheme because they allow a single user to give multiple coins to another, inflating their reputation. In our scheme, it is possible to prove that a collection of votes came from *distinct* users. This ability to prove distinctness while maintaining the mutual anonymity of both voters and vote receivers is the key technical achievement of our construction.

Schemes for anonymous credentials [28, 10, 8, 25] employ some similar techniques to those of our constructions and share analogous goals. In fact, our signatures of reputation might be considered a type of anonymous credential. There are two key distinctions, however. First, proposals for anonymous credentials typically concern the setting of access control based on trust derived from explicit authorities, whereas this work aims to support trust derived from a very different source: the aggregate opinions of other users. Second, like e-cash based approaches, existing anonymous credential schemes lack a mechanism for proving that votes or credentials come from distinct users while simultaneously hiding the identities of those users.

Our setting superficially resembles that of e-voting protocols [47, 46, 37], in that it allows the casting of votes while maintaining certain privacy properties. However, schemes for e-voting are designed for an election scenario in which the candidates have no need to receive votes and prove possession of votes anonymously, among other differences, and cannot be used to achieve the properties we require.

Private stream searching. The problem of private stream searching is essentially a variant of single-database, computationally-private information retrieval [33, 59, 24, 31], which

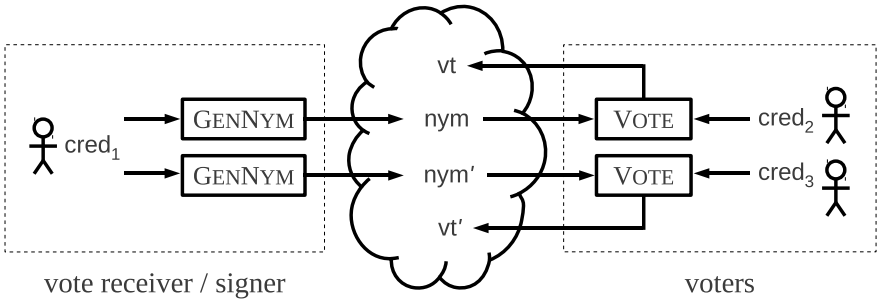


Figure 1.1: A user anonymously posts two one-time pseudonyms, each of which receives a vote.

is in turn closely related to oblivious transfer [70, 62]. The key difference between traditional PIR and private stream searching is that the former requires communication dependent on the size of the entire database rather than the size of the portion retrieved. In some streaming settings, a private searching scheme with communication independent of the size of the stream or database is desirable. Another difference between the PIR and private search settings is the fact that most PIR constructions model the database to be searched as a long bitstring and the queries as indices of bits to be retrieved. In contrast, both our scheme and that of Ostrovsky and Skeith allow queries based on a search for keywords within text. Both may also retrieve pieces of data by index, however. The text associated with a block of data in the database against which queries are matched is arbitrary, so by including strings of the form "blocknumber:1," "blocknumber:2," ... in the text associated with each block of data, they may be explicitly retrieved by appropriate queries. There has been some consideration of constructions supporting retrieval by keyword rather than block index in the PIR literature [32, 58, 43], but none of these systems has communication dependent only on the size of the data retrieved rather than some function of the length of the database or stream.

We also note that private searching may seem related to the problem of searching on encrypted data [80, 17, 45]. At a technical level, however, the two actually bear little resemblance. When searching on encrypted data, the data is hidden from the server, but private searching requires that the client's queries remain hidden and places no such requirement on the data.

1.3 Signatures of Reputation

Having surveyed related work, in this section we continue to describe the new techniques we have developed. The block diagrams of Figures 1.1 through 1.3 illustrate the flow of information between the algorithms of our schemes for signatures of reputation and private stream searching. As explained previously, the construction we have developed for signatures of reputation supports monotonic measures of reputation, and we call the units of such

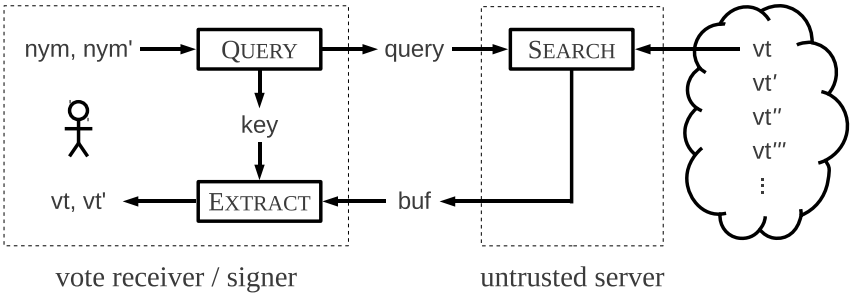


Figure 1.2: Eventually, the user shown in Figure 1.1 retrieves the votes by constructing an encrypted query for the corresponding pseudonyms.

reputation "votes." In the following discussion, we refer to each user as a *vote receiver*, *voter*, *signer*, or *verifier* depending on their role in the specific algorithm being discussed.

To ensure *receiver anonymity*, a vote receiver invokes the GENNYM algorithm to compute a "one-time pseudonym" called a nym, which they attach to some content that they publish and wish to receive credit for. A voter can then use the VOTE algorithm on a nym to produce a vote which hides their identity, even from the recipient (referred to as *voter anonymity*). The voter posts the resulting vote on a public server. This voting process is shown in Figure 1.1. To retrieve votes cast for their pseudonyms, a vote receiver performs a private search, as shown in Figure 1.2. The algorithms involved in this process (QUERY, SEARCH, and EXTRACT) will be discussed in the next section. After collecting some votes, a signer runs the SIGNREP algorithm on a given message to construct a signature of reputation, which must not reveal the signer's identity (*signer anonymity*). We also ensure that a malicious signer cannot inflate its reputation (*reputation soundness*). Figure 1.3 illustrates the signing and verification process.

Reputation semantics. Before further describing each of these algorithms, several points should be made regarding the interpretation of reputation within this context. First of all, any system that allows users to increase each other's reputations at will must somehow limit its membership if it is to remain meaningful—otherwise malicious users could obtain unbounded reputation by simply creating additional identities (a Sybil attack [41]). A mechanism to prevent or mitigate such attacks is necessary regardless of any privacy properties a reputation system may provide.[1] Since this aspect of devising a useful reputation system is not the focus of this work, we simply assume the existence of a party termed the *registration authority* (RA) which represents the system's mechanism for limiting membership. To participate in the system, each user must have credentials issued by RA which certify the user as a legitimate member.

Although we describe the RA as a single party which generates user credentials, it can

[1]For example, today's services often address this issue by requiring a (previously unused) mobile phone number upon registration, to which a validation code is sent via SMS.

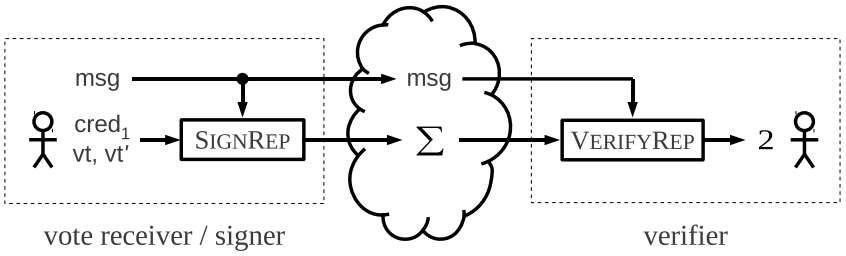


Figure 1.3: A user signs a message while proving they have votes from two distinct users.

be distributed amongst multiple parties like the key generating server of typical identity-based encryption (IBE) schemes [44]. While our construction requires trust in the RA for both privacy and reputation soundness, it need only be trusted while generating credentials and may thereafter go offline. After registering a user, it plays no further role in storing or managing their reputation—in contrast to systems based on an online trusted party. Note that, in our scheme, the RA is incapable of regenerating a user's credential once it destroys the randomness used to produce it. In this respect, users need not trust the RA as much as an IBE key generator, which can regenerate private keys at any point. Admittedly, the distinction between a party which is always honest and one which is only honest for some initial period is subtle, but it is an important difference in some real scenarios. For example, in the case of an honest RA whose servers are eventually confiscated by a law enforcement agency, the users registered prior to that point could continue using the system indefinitely without risking their privacy. The privacy of all users of an IBE scheme would be compromised in that scenario. Nevertheless, devising a scheme which maintains privacy despite an initially malicious RA is an important problem for future work. On the other hand, relying on the honesty of the RA for reputation soundness seems inevitable, since a malicious RA could always register phony users to create votes and inflate reputations.

At this point, one might raise the concern that, if each user has received a unique number of votes, the reputation value itself is identifying. Clearly, there is an inherent tradeoff between the precision of a measure of reputation and the anonymity of a user with any specific value, as pointed out by Steinbrecher [76]. The solution is to use a sufficiently coarse-grained reputation. In our construction, a user may prove any desired lower bound on their reputation instead of revealing the actual value; this is accomplished by simply omitting some votes when invoking the SignRep algorithm. In this way, our construction allow users to implement their own policies for the precision of their reputations. For example, one policy would be to always round down to a power of two.

Another issue to consider is the connection between a piece of content a user has posted and the attached nym. Two abuses are possible: reposting the nym of another user with a piece of undesirable content in order to malign the user's reputation and reposting the desirable content of another user with one's own nym in order to steal the credit. The former problem can be easily prevented by including a signature within the nym linking it to a

specific message. However, this is only useful in a reputation system supporting negative feedback. Since our constructions only support monotonic reputation, we do not include this feature. On the other hand, there is, in general, no simple way of preventing the latter problem. Note that the problem of assigning credit does not stem from the anonymity that we provide; it equally affects non-privacy-preserving reputation systems. In the case of audio or video content, one way to address this would be to use digital watermarking techniques to embed the nym throughout the content [36]. Other approaches could be based on a public timestamping service.

Algorithms and properties. With those considerations covered, we are ready to detail the set of algorithms that constitutes a scheme for signatures of reputation and consider the properties we require of each. All of the below except VerifyRep may be randomized.

Setup(1^λ) $\to$ (params, authkey): The Setup algorithm is run once on security parameter 1^λ to establish the public parameters of the system params and a key authkey for the registration authority.

GenCred(params, authkey) $\to$ cred: To register a user, the registration authority runs GenCred and returns the user's credential cred.

GenNym(params, cred) $\to$ nym: The GenNym algorithm produces a one-time pseudonym nym from a user's credential.

Vote(params, cred, nym) $\to$ vt or $\perp$: Given the credentials cred of some user and a one-time pseudonym nym, Vote outputs a vote from that user for the owner of nym, or $\perp$ in case of failure (e.g., if nym is invalid).

SignRep(params, cred, V, msg) $\to$ Σ or $\perp$: Given the credentials cred of some user, the SignRep algorithm constructs a signature of reputation Σ on a message msg using a collection of c votes $V = \{\mathsf{vt}_1, \mathsf{vt}_2, \ldots, \mathsf{vt}_c\}$ for that user. The signature corresponds to a reputation $c' \leq c$, where c' is the number of distinct users who generated votes in V. The SignRep algorithm outputs $\perp$ on failure, specifically, when V contains an invalid vote or one whose recipient is not the owner of cred.

VerifyRep(params, msg, Σ) $\to$ c or $\perp$: The VerifyRep algorithm checks a purported signature of reputation on msg and outputs the corresponding reputation c, or $\perp$ if the signature is invalid.

In the following chapter, we provide fully rigorous definitions for the privacy and security properties we aim to enforce. Here, we introduce them at an intuitive level and discuss some of the subtleties in defining them appropriately.

First, we would like to ensure that a user may produce signatures of reputation anonymously. Furthermore, it should be impossible to determine whether two different signatures were produced by the same user. This may be defined by the following game.

The challenger begins by generating the public parameters and a list of user credentials $\mathsf{cred}_1, \ldots, \mathsf{cred}_n$. An adversary $\mathcal{A}$ is given access to all the credentials and may use them to generate pseudonyms and votes before eventually printing a message msg, two indices $i_0, i_1 \in \{1, \ldots, n\}$, and two sets of votes V_0, V_1. The challenger flips a coin $b \in \{0, 1\}$ and returns $\Sigma_b = \textsc{SignRep}(\mathsf{params}, \mathsf{cred}_{i_b}, V_b, \mathsf{msg})$ to $\mathcal{A}$, which prints a guess b'. We say that $\mathcal{A}$ has won the game if $b = b'$ and $\textsc{VerifyRep}(\mathsf{params}, \mathsf{msg}, \Sigma_0) = \textsc{VerifyRep}(\mathsf{params}, \mathsf{msg}, \Sigma_1)$. That is, the value of b should affect neither the reputation values of the resulting signatures nor their validity. If the advantage (that is, the probability of winning the game minus one-half) of every probabilistic, polynomial time (PPT) adversary $\mathcal{A}$ is negligible in the security parameter, we say that the scheme is signer anonymous.

Complementing the ability to produce a signature of reputation anonymously is the ability to receive the necessary votes anonymously. In this case, we require that a pseudonym generated by the GenNym algorithm reveal nothing about its owner in the absence of that user's credential. An adversary $\mathcal{A}$ playing the corresponding game will select two users $1 \leq i_0, i_1 \leq n$ and must guess which produced the challenge $\mathsf{nym}^* = \textsc{GenNym}(\mathsf{params}, \mathsf{cred}_{i_b})$. Since we allow users to identify their own pseudonyms, we cannot provide all the credentials to $\mathcal{A}$ in this case. Instead, we provide $\mathcal{A}$ with access to an oracle which will reveal individual credentials on demand (a "corrupt" query) or use them to produce pseudonyms, votes, and signatures as requested. Then, to win the game, we require that $\mathcal{A}$ not corrupt either i_0 or i_1. We also require that $\mathcal{A}$ not request a signature from i_0 or i_1 using a vote that was cast for nym^*, since the reply would immediately reveal b (the signer is i_b if and only if the reply is not $\perp$). If the advantage of every PPT $\mathcal{A}$ in this game is negligible in the security parameter, the scheme is receiver anonymous.

Astute readers may note that we have not properly defined what it means for a vote to have been "cast for nym^*," since we have no information about how the adversary may have constructed it. To resolve this issue, the full definitions in the following chapter include "opening" algorithms which reveal the creator of a pseudonym and the voter and recipient of a given vote. To operate, they require a special opening key which may be generated during setup, just as in group signature schemes. However, while this tracing is an explicit feature of group signatures, here we use it only to establish a "ground truth" for definitional purposes. In an actual implementation, the opening key would not be generated.

We wish to define the next privacy property, voter anonymity, to encompass the strongest form of unlinkability compatible with the general semantics of the scheme, as we did in the case of receiver anonymity. Doing so is more subtle in this case, however, due to the necessity of detecting duplicate votes. Because we require a SignRep algorithm to demonstrate the number of votes from distinct users, such an algorithm can be used by a vote receiver to determine whether two votes cast for any of their pseudonyms were produced by the same voter (duplicates). That is, the receiver can try to use the two votes to produce a signature and then check the reputation of the result with VerifyRep.

In defining voter anonymity, we allow precisely this type of duplicate detection, but nothing more. While initially this may seem like an "exception" to the unlinkability of

votes, in actuality, it is not only inevitable,[2] but also unlikely to be a practical concern. Although a vote receiver must be able to detect duplicate votes, we can still avoid the voting histories we aim to eliminate. In particular, our definition ensures that in the following cases it is not possible to determine whether two votes were cast by the same user (i.e., to link the votes):

1. A user cannot link a vote for one of their pseudonyms with a vote for a pseudonym of another user, nor can they link two votes for distinct pseudonyms of another user (or two different users).
2. A colluding group of users cannot link votes between their pseudonyms, provided the pseudonyms correspond to different credentials. Furthermore, they are not able to link the numbers of duplicates they have observed. For example, if a user determines that they have received two votes from one user and three votes from another, they will have no way of matching these totals up with those of another colluding user.

In the corresponding game, $\mathcal{A}$ selects two indices $i_0, i_1 \in \{1, \ldots, n\}$ and nym and is given $\mathsf{vt}^* = \textsc{Vote}(\mathsf{params}, \mathsf{cred}_{i_b}, \mathsf{nym})$ as a challenge. As before, they are given access to the oracle and must make a guess b'. In this case, we require that if $\mathcal{A}$ requests through the oracle that the user corresponding to nym produce a signature using vt^*, then votes from both i_0 and i_1 must be included. Otherwise, the number of distinct votes in the resulting signature would directly reveal b. Additionally, we disqualify $\mathcal{A}$ if they both corrupt the user corresponding to nym and request a vote on nym from either i_0 or i_1. This is necessary because the status of such a vote as a duplicate of vt^* (or lack thereof) would reveal b. If every PPT $\mathcal{A}$ has negligible advantage in this game, the scheme is voter anonymous.

To define the soundness of a scheme for signatures of reputation, we use a computational game in which an adversary $\mathcal{A}$ must forge a signature of reputation Σ on some message msg. We disqualify $\mathcal{A}$ if Σ was the reply to one of its oracle queries, and we require that Σ have reputation strictly greater than what it could have if the adversary had used the scheme normally. The value of the best such legitimately obtainable reputation will depend on several things: the number of users the adversary has corrupted (since the adversary may use their credentials to produce votes), the number of votes received from honest users via oracle queries, and how those votes were distributed amongst the corrupted users. More precisely, let ℓ_1 be the *number of corrupted users* and ℓ_2 be the *greatest number of distinct honest users that voted for a single corrupt user*. Then we require that $\textsc{VerifyRep}(\mathsf{params}, \mathsf{msg}, \Sigma) > \ell_1 + \ell_2$ for the adversary to succeed. If, for every PPT $\mathcal{A}$, the probability of winning this game is negligible in the security parameter, then the scheme is sound.

In some applications, a weaker version of soundness may suffice and may be desirable for greater efficiency. One natural way to relax the definition is to specify an additional

[2]Allowing proofs of vote distinctness while eliminating the ability to identify duplicates could only be possible if the notion of discrete votes is abandoned. This approach would require *all* votes in the system to be aggregated into a indivisible block before they can be used to produce signatures, a vastly impractical solution.

security parameter $0 \leq \varepsilon < 1$ as a multiplicative bound on the severity of cheating we wish to prevent. Specifically, we say that a scheme is *ε-sound* if a signature of reputation c convinces the verifier that the signer must possess at least $(1-\varepsilon) \cdot c$ valid votes. This can be defined as above, but using the requirement that $(1-\varepsilon) \cdot \textsc{VerifyRep}(\mathsf{params}, \mathsf{msg}, \Sigma) > \ell_1 + \ell_2$.

Highlights of our construction. The following chapter details our construction for signatures of reputation and proves that it satisfies all the properties just discussed. Here, we briefly survey some of the scheme's technical features and underlying ideas.

Our scheme for signatures of reputation relies on a bilinear map (symmetric or asymmetric) between prime order groups and can produce sound signatures of reputation c of size $O(c)$ or ε-sound signatures of size $O(\frac{1}{\varepsilon} \log c)$. The proofs of the privacy and security properties are based on the relatively standard SDH and decision linear assumptions, BB-HSDH and BB-CDH [8], and a new constant-size, non-interactive, computational assumption called SCDH, which we prove hard in generic groups. Additionally, the ε-sound variant of our scheme requires the random oracle model.

Throughout our construction, we make extensive use of the Groth-Sahai scheme for non-interactive zero-knowledge (NIZK) proofs [49], which can be used to efficiently demonstrate possession of signatures, ciphertexts, and their relationships while maintaining unlinkability properties. One unique (to our knowledge) feature of our construction is the use of *nested NIZKs*, that is, NIZKs which prove knowledge of other NIZKs and demonstrate that they satisfy the verification equations. This situation arises because a user's credentials contain a signature from the registration authority, and a user includes a NIZK proof of the validity of this signature when they cast a vote. When a signer later uses the vote, they include this NIZK within a further NIZK to demonstrate the validity of the votes while maintaining signer anonymity.

We give signers the ability prove the distinctness of their votes through the following mechanism. Each user credential contains, among other components, a "voter key" v and a "receiver key" r. A valid vote must contain a certain deterministic, injective function of these keys: $f(v, r)$. Thus, duplicate votes can be detected when $f(v_1, r) = f(v_2, r)$. To receive votes anonymously, a user includes in each nym an encryption of their receiver key $E(r)$ under their own public key. Using a homomorphism, the voter uses this ciphertext to compute $E(f(v, r))$ and places it within the vote; later, the receiver decrypts this to obtain $f(v, r)$. To maintain signer anonymity when using a series of values $U_1 = f(v_1, r)$, $U_2 = f(v_2, r)$, ... to sign a message, the signer blinds them with a (single) exponent to produce a list $U_1^s, U_2^s, \ldots$, which is included in the signature of reputation along with proof of knowledge of the exponent. Note that $U_1^s, U_2^s, \ldots$ will be distinct if and only if $U_1, U_2, \ldots$ are.

To reduce the size of the signatures, we employ a sampling technique. Specifically, we can achieve ε-soundness while only including a random subset of the votes of size $O(\frac{1}{\varepsilon})$, independent of the original number of votes. To ensure the sample is random, we require the signer

to first commit to the entire list of votes, then use the commitment as a challenge specifying which must be included. To efficiently demonstrate that the correct votes were included, we compute the commitment using a Merkle hash tree [67] and include the corresponding off-path hashes with each vote, resulting in a final signature of size $O(\frac{1}{\varepsilon} \log c)$.

1.4 Private Stream Searching

We now turn to the other primary set of techniques we have developed: improved constructions for private stream searching. We begin by defining the algorithms of a scheme for private stream searching and discussing the original scheme due to Ostrovsky and Skeith before giving an overview of our techniques.

To use a scheme for private stream searching, a client creates an encrypted query for the set of search terms that they are interested in, then sends the encrypted query to a server, as shown in Figure 1.2. While the figure depicts retrieval of the cryptographic votes necessary to construct signatures of reputation, our construction can also be used to obtain any content of interest. After receiving the client's query, the server runs a search algorithm on a stream of files while keeping an encrypted buffer storing information about files for which there is a match. The encrypted buffer is periodically returned to the client to enable the client to reconstruct the files that have matched its query. The key aspect of a private searching scheme is that a server is capable of conducting the search even though it does not know which search terms the client is interested in or which files match them. Formally, we define such a scheme as consisting of the following three algorithms.

$\textsc{Query}(1^\lambda, \varepsilon, S, m, n) \to (\mathsf{query}, \mathsf{key})$: The Query algorithm is run by the client to construct an encrypted query. It takes a security parameter 1^λ, a correctness parameter ε, a set of search terms $S \subset \{0,1\}^*$, an upper bound m on the number of files to retrieve, and an upper bound n on their total length. From these it produces an encrypted query and corresponding key.

$\textsc{Search}(\mathsf{query}, f, \mathsf{buf}) \to \mathsf{buf}'$: After the receiving an encrypted query from the client, the server runs the Search algorithm to evaluate the query on each file $f \in \{0,1\}^*$ in the stream, updating a buffer of encrypted results buf. For the first file, the server sets $\mathsf{buf} = \perp$; for each subsequent file, the server uses the value returned by the previous iteration. At any point, the buffer of results may be returned to the client.

$\textsc{Extract}(\mathsf{key}, \mathsf{buf}) \to F$: The Extract algorithm is then run by the client to extract the files which matched the query from buf using key. It outputs the set of matching files $F = \{ f \,|\, S \cap \mathsf{words}(f) \neq \varnothing \}$, where words is a function that returns the set of keywords associated with a file.[3]

[3]This function will vary by application. For searching on text, it may simply split on whitespace to return all the words in the file. For binary files, it may return associated metadata or all sequences of bytes the

We define privacy for a private stream searching scheme with the following game. The adversary gives two sets of search terms S_0 and S_1 to the challenger, who then flips a coin $b \in \{0, 1\}$, runs $\text{QUERY}(1^\lambda, \varepsilon, S_b, m, n)$, and returns query to the adversary. The adversary then outputs a guess b' and wins if $b = b'$. We say that a private searching scheme is *semantically secure* if every PPT $\mathcal{A}$ has negligible advantage in this game.

There are a few comments to be made about the parameters m, n, and ε. First, note that the security definition for private stream searching necessitates that the server return the same amount of data regardless of which files (if any) matched the query. If this were not the case, the server could easily mount a dictionary attack using the SEARCH algorithm to determine the exact query keywords. As a result, any scheme for private stream searching requires an a priori upper bound on the number of files to retrieve (if we assume fixed-length files) or their total length. If more files match or they are too long, the scheme will fail to return them. Furthermore, all existing private stream searching schemes (ours and Ostrovsky-Skeith) have the possibility of random failures in which not all matching files are recoverable, even if the bounds m and n are satisfied. We include the correctness parameter ε to specify an upper bound on the probability of such a failure and require that a scheme use redundancy or other means to achieve the corresponding level of reliability.

The Ostrovsky-Skeith scheme. Our scheme is best understood in relation to the original one provided by Ostrovsky and Skeith in the paper which proposed the problem of private stream searching, so we now review its basic idea [72]. Their scheme can use any additively homomorphic public key cryptosystem, and they suggest Paillier in particular [74, 38]. First, a public dictionary of keywords D is fixed; only strings within this set may be used as search terms. To construct a query for the disjunction[4] of a set of search terms $S \subseteq D$, the user generates a new key pair and produces an array of ciphertexts, one for each $w \in D$. If $w \in S$, a one is encrypted; otherwise a zero is encrypted. A server processing a document in its stream then computes the product of the array entries corresponding to the keywords found in the document. This will result in the encryption of some value c, which, by the homomorphism, is non-zero if and only if the document matches the query. The server may then in turn compute $E(c)^f = E(cf)$, where f is the content of the document, obtaining either an encryption of (a multiple of) the document or an encryption of zero.

Ostrovsky and Skeith propose the server keep a large array of ciphertexts as a buffer to accumulate matching documents; each $E(cf)$ value is multiplied into a number of random locations in the buffer. If the document matches the query then c is non-zero and copies of that document will be placed into these random locations; otherwise, $c = 0$ and this step will add an encryption of 0 to each location, having no effect on the corresponding plaintexts. A

file contains if the ability to search the binary content itself is desired. In any case, words is assumed to be public or specified in the clear within query.

[4]They also mention an extension allowing a single conjunction by using the BGN cryptosystem rather than Paillier [18]. This extension can also be applied to our scheme in the same way.

fundamental property of their solution is that if two different matching documents are ever added to the same buffer location, a collision will result and both copies will be lost. If all copies of a particular matching document are lost due to collisions then that document is lost, and when the buffer is returned to the client, they will not be able to recover it.

To avoid the loss of data in this approach one must make the buffer sufficiently large so that this event does not happen too often. This requires that the buffer be much larger than the number of documents expected to match. In particular, Ostrovsky and Skeith show that a given probability ε of successfully obtaining all matching documents may be obtained with a buffer of size $O(n \log n)$,[5] where n is the upper bound on the total length of the matching documents. While effective, this scheme results in inefficiency due to the fact that a significant portion of the buffer returned to the user consists of empty locations and document collisions.

Unfortunately, our experiments have shown that this source of inefficiency is indeed substantial. The Ostrovsky-Skeith paper did not specify explicitly state a minimum buffer length for a given number of files expected to be retrieved and a desired probability of success, but instead gave a loose upper bound on the length. To determine the efficiency of their scheme more precisely, we ran a series of simulations to determine exactly how small the buffer could be made with $\varepsilon = \frac{1}{20}$. The results are given in Chapter 4; in short, the data returned is approximately fifty times as large as the files being retrieved.

Improving efficiency. We now survey the ideas behind our scheme and the improvements obtained as a result. Our primary result is a set of techniques for improving space efficiency. Rather than using a large buffer and attempting to avoid collisions, we allow collisions and recover from them using additional information. Each matching document in our system is copied randomly over approximately half of the locations in the buffer. A pseudo-random function g (for which both client and server have the key) will determine pseudo-randomly with probability $\frac{1}{2}$ whether the document is copied into a given location, where the function takes as inputs the document number (document number i is the ith document seen by the server) and buffer location. While any one particular buffer location will not likely contain sufficient information to reconstruct any one matching document, with high probability all the information from all the matching documents can be retrieved from the whole system by the client given that the client knows the number of matching documents and that the number of matching documents is less than the buffer size. The client can do this by decrypting the buffer and then solving a linear system to retrieve the original documents.

To do so, the client must obtain a list of the indices of the documents in the stream which matched the query. We describe two ways of accomplishing this. The first method (referred to as the simple metadata construction) is based on the original Ostrovsky-Skeith construction. To employ the alternative method (referred to as the Bloom filter construc-

[5]Specifically, they define a correctness parameter γ and use a buffer of size $O(\gamma n)$. They then show that a given success probability may be achieved with a γ that is $O(\log n)$.

Private stream searching scheme	Storage and comm.	Client reconstruction time
Ostrovsky-Skeith 2005	$O(n \log n)$	$O(n \log n)$
Our scheme (simple metadata)	$O(n + m \log m)$	$O(n + m^{2.376})$
Our scheme (Bloom filter)	$O(n + m \log \frac{t}{m})$	$O(n + m^{2.376} + t \log \frac{t}{m})$

Table 1.1: The two variants of the new scheme retrieve m documents while incurring only linear overhead in terms of the total length n.

tion), the server maintains an encrypted Bloom filter that efficiently keeps track of which document numbers were matched. The Bloom filter construction provides a compact way of representing the set indices of matching documents and requires less space than the simple metadata construction under some circumstances.

These techniques can improve the space efficiency asymptotically in some situations. Specifically, our scheme achieves the optimal linear communication from the server to the client and server storage overhead in returning the content of m matching documents with total length n. To return the necessary metadata (primarily the indices of the matching documents), we may use the original scheme with $O(m \log m)$ communication and storage. When considering unit length (i.e., one 1024-bit group element) documents, m equals n and our scheme then shares the same overall $O(n \log n)$ communication complexity as Ostrovsky-Skeith. However, because our scheme decouples the logarithmic communication factor from the document length, we improve the communication complexity for longer documents. Using the Bloom filter construction for returning the metadata requires $O(m \log \frac{t}{m})$ communication and storage, where t is the total number of documents searched. A disadvantage of this technique is a step in reconstructing the matching documents on the client with $O(t \log \frac{t}{m})$ time complexity, introducing a dependency on the overall stream length. However, this step consists only of computing a series of hash values, which is greatly outweighed by other costs in practice. These efficiency improvements and tradeoffs are summarized in Table 1.1. While the asymptotic efficiency improvements are perhaps marginal, the greatest benefits are obtained in the practical performance. In Chapter 4, we will see that the new scheme incurs overheard of about a factor of three, an order of magnitude improvement.

Other improvements. Our scheme for private stream searching is also more flexible than the previous work in two important ways. First, we remove the need for the predetermined set of possible search terms D. This is crucial because many of the strings a user may want to search for are obscure (e.g., names of particular people or other proper nouns) and including them in D would already reveal too much information. Since the size of encrypted queries is proportional to $|D|$, it may not be feasible to fill D with, say, every person's name, much less all proper nouns. In the case of our signatures of reputation, retrieving votes based on the corresponding one-time pseudonyms requires searching for arbitrary binary strings and would be impossible using a fixed dictionary.

We remove this limitation through a simple hash table based technique that allows any string to be a search term. In QUERY, we create a new key pair as before, then create an ℓ-element array of ciphertexts initialized to encryptions of zero, where ℓ is selected based on the correctness parameter ε. Next, we hash each search term $s \in S$ and set the corresponding element $(h(s) \bmod \ell)$ of the array to an encryption of one. To process a file in SEARCH, the server then simply hashes its associated keywords and computes the product of the corresponding entries in the hash table. The disadvantage of this approach is the possibility of false positive matches due to collisions within the hash table, which cause files to be erroneously retrieved. This possibility must be taken into account in relation to the correctness parameter ε and the bound m on the number of matching files.

Another significant improvement we have made to the flexibility of private stream searching is the addition of support for variable length documents. Previously, only fixed length blocks had been considered, which requires setting an upper bound on the document length and (inefficiently) treating all documents as if they are that long. In our scheme, we allow the client to set separate bounds on the number of matching documents and their total length. Provided both bounds are satisfied, the data may be distributed arbitrarily.

1.5 Summary

To recap this chapter, the cryptographic tools described in the previous section can be used to create a new type of privacy preserving online identity in the following manner. Rather than creating separate accounts for various applications under their real name or pseudonyms, users would instead use a single set of cryptographic credentials. A registration authority of some sort is necessary to prevent users from obtaining an unlimited number of identities but would otherwise play no role in managing a user's identity. With their credentials, a user could publish information anonymously and unlinkably, attaching a new one-time pseudonym to each post and only revealing their name or a long-term pseudonym when specifically desired. To receive credit for the information they post, a user would register queries for their one-time pseudonyms with the servers that host the applications they use. These queries would be used to periodically return votes others cast for the posts and any additional information, such as the content of replies to the posts. Our new scheme for private stream searching allows this to take place without linking the user to the one-time pseudonyms or the pseudonyms to each other. Using our scheme for signatures of reputation, the user could then sign further posts with a proof demonstrating a lower bound on the number of votes collected so far, giving the published information much of the credibility offered by more explicit forms of identity.

In the following two chapters, we detail our constructions for signatures of reputation and private stream searching. After that, the next two chapters explore the practical feasibility of our proposals using current technology and the possibility of further attacks on anonymity if they were to be used. Finally, we conclude in Chapter 6 by surveying areas of future work.

Chapter 2

Constructing Signatures of Reputation

In this chapter, we provide the full details of our proposed signatures of reputation. We begin with rigorous definitions for the properties we desire and an explanation of technical tools from which our scheme is devised in Sections 2.1 and 2.2. In Section 2.3, we describe the algorithms of our construction, and in Section 2.4, we explain how they can be modified to reduce space requirements. The reader may wish to briefly review the list of algorithms in Section 1.3 before continuing in this chapter.

2.1 Properties

The most basic property required of a construction for signatures of reputation is correctness: the algorithms should produce the expected results when executed normally. We define this property as follows.

Definition 1 (Correctness for signatures of reputation)**.** *Let n be a positive integer and $S \subseteq \{1, \ldots, n\}$. Set* $(\mathsf{params}, \mathsf{authkey}) \leftarrow \textsc{Setup}(1^\lambda)$ *and* $\mathsf{cred}_i \leftarrow \textsc{GenCred}(\mathsf{params}, \mathsf{authkey})$*, for* $i \in \{1, \ldots, n\}$*. Set* $\mathsf{nym} \leftarrow \textsc{GenNym}(\mathsf{params}, \mathsf{cred}_1)$*,* $V = \{\ \mathsf{vt} \mid \mathsf{vt} \leftarrow \textsc{Vote}(\mathsf{params}, \mathsf{cred}_i, \mathsf{nym}), i \in S\ \}$*, and* $\Sigma \leftarrow \textsc{SignRep}(\mathsf{params}, \mathsf{cred}_1, V, \mathsf{msg})$*, for some message* msg*. If the preceding implies, with probability one, that* $\textsc{VerifyRep}(\mathsf{params}, \mathsf{msg}, \Sigma) = |S|$*, then we say that the scheme is* correct*.*

In the preceding chapter, we gave intuitive descriptions of the four intended privacy and security properties: receiver anonymity, voter anonymity, signer anonymity, and reputation soundness. Defining these properties rigorously requires considerably more subtlety. In particular, to ensure our definitions are well-formed, we require the existence of several additional algorithms. Given a special "opening key" produced by an alternate version of Setup, the two opening algorithms (which must be deterministic) reveal the users associated

with pseudonyms and votes, thereby establishing a ground truth to which we can refer when defining the privacy and security properties. These are directly analogous to the opening algorithm of group signature schemes. However, while opening is considered an intended feature of group signatures, in our case the opening algorithms exist solely for the definitions and would not be used in practice. For brevity, each of the opening algorithms is given params and a list of user credentials $\mathsf{cred}_1, \ldots, \mathsf{cred}_n$ as implicit arguments.

$\textsc{Setup}'(1^\lambda) \to (\mathsf{params}, \mathsf{authkey}, \mathsf{openkey})$: $\textsc{Setup}'$ produces values $\mathsf{params}, \mathsf{authkey}$ according to the same distribution as $\textsc{Setup}$, but also outputs an opening key $\mathsf{openkey}$.

$\textsc{OpenNym}(\mathsf{openkey}, \mathsf{nym}) = i$: Output the index of the credential that produced nym.

$\textsc{OpenVote}(\mathsf{openkey}, \mathsf{vt}) = (i, \mathsf{nym})$: Output the index of the credential and the nym from which the vote vt was constructed.

Any scheme for signatures of reputation must provide an implementation of the above algorithms that is *correct* according to the following definition.

Definition 2 (Correctness of opening algorithms)**.** *Let n be a positive integer and set $(\mathsf{params}, \mathsf{authkey}, \mathsf{openkey}) \leftarrow \textsc{Setup}'(1^\lambda)$ and $\mathsf{cred}_i = \textsc{GenCred}(\mathsf{params}, \mathsf{authkey})$ for $i \in \{1, \ldots, n\}$. Then given any $i \in \{1, \ldots, n\}$ and $\mathsf{nym} \leftarrow \textsc{GenNym}(\mathsf{params}, \mathsf{cred}_i)$, we require that $\textsc{OpenNym}(\mathsf{openkey}, \mathsf{nym}) = i$. In addition, given any $i, j \in \{1, \ldots, n\}$, $\mathsf{nym} \leftarrow \textsc{GenNym}(\mathsf{params}, \mathsf{cred}_i)$, and $\mathsf{vt} \leftarrow \textsc{Vote}(\mathsf{params}, \mathsf{cred}_j, \mathsf{nym})$, we require that $\textsc{OpenVote}(\mathsf{openkey}, \mathsf{vt}) = (j, \mathsf{nym})$. If both of these properties hold, we say that the scheme has* correct opening algorithms.

For later notational convenience, given a set of votes V, we also define the functions $\textsc{OpenVoters}(\mathsf{openkey}, V) = \{\ i \mid (i, \mathsf{nym}) = \textsc{OpenVote}(\mathsf{openkey}, \mathsf{vt}), \mathsf{vt} \in V\ \}$ and $\textsc{OpenReceivers}(\mathsf{openkey}, V) = \{\ \mathsf{nym} \mid (i, \mathsf{nym}) = \textsc{OpenVote}(\mathsf{openkey}, \mathsf{vt}), \mathsf{vt} \in V\ \}$.

Before describing the games defining each of the privacy and security properties, one more preliminary matter must be discussed. In the games we define, the adversary may make queries to an oracle $\mathcal{O}$. The oracle is given access to a list of user credentials $\mathsf{cred}_1, \ldots, \mathsf{cred}_n$ and responds to the following four types of queries. On input $(\text{“corrupt”}, i)$, $\mathcal{O}$ returns cred_i. On input $(\text{“nym”}, i)$, $\mathcal{O}$ returns $\mathsf{nym} \leftarrow \textsc{GenNym}(\mathsf{params}, cred_i)$. On input $(\text{“vote”}, i, \mathsf{nym})$, $\mathcal{O}$ returns $\mathsf{vt} \leftarrow \textsc{Vote}(\mathsf{params}, \mathsf{cred}_i, \mathsf{nym})$. On input $(\text{“signrep”}, i, V, \mathsf{msg})$, $\mathcal{O}$ returns $\Sigma \leftarrow \textsc{SignRep}(\mathsf{params}, \mathsf{cred}_i, V, \mathsf{msg})$. In each case, $\mathcal{O}$ also logs the tuple on which it was queried and its response by adding them to a set L. We will refer to the logged queries and responses in order to state the winning conditions for each game.

Receiver anonymity. The receiver anonymity property captures the notion that a one-time pseudonym generated by the $\textsc{GenNym}$ algorithm should reveal nothing about its owner, unless the adversary has seen that user's credential or made a $\textsc{SignRep}$ query which trivially

reveals the owner. This property may be defined by the following game, where st denotes the internal state of the adversary.

$$\mathbf{Pr}^{\mathsf{RANON}}_{\mathcal{A}}(\lambda) = \Pr\left[\begin{array}{c|c} b = b' \ \wedge \\ \textsc{Legal}(i_0^*, i_1^*, \mathsf{openkey}, L) & \begin{array}{c} (\mathsf{params}, \mathsf{authkey}, \mathsf{openkey}) \leftarrow \textsc{Setup}'(1^\lambda); \\ [\mathsf{cred}_i \leftarrow \textsc{GenCred}(\mathsf{params}, \mathsf{authkey})]_{1 \le i \le n}\,; \\ (i_0^*, i_1^*, \mathsf{st}) \leftarrow \mathcal{A}^{\mathcal{O}_L}(\mathsf{params});\ b \xleftarrow{R} \{0,1\}; \\ \mathsf{nym}^* \leftarrow \textsc{GenNym}(\mathsf{params}, \mathsf{cred}_{i_b^*});\ b' \leftarrow \mathcal{A}^{\mathcal{O}_L}(\mathsf{st}, \mathsf{nym}^*) \end{array} \end{array}\right]$$

To prevent $\mathcal{A}$ from winning this game through normal usage of the scheme, we make the following requirements on its queries and challenge (i_0^*, i_1^*), which we abbreviate as "Legal." First, $(\text{"corrupt"}, i_0^*) \notin L$ and $(\text{"corrupt"}, i_1^*) \notin L$. In other words, if an adversary must compromise a user's private credentials to detect their pseudonyms, we will not consider that a failure of receiver anonymity.[1] Second, for all $(\text{"signrep"}, i, V, \mathsf{msg}) \in L$, if $i \in \{i_0^*, i_1^*\}$, we require that $\mathsf{nym}^* \notin \textsc{OpenReceivers}(\mathsf{openkey}, V)$. An adversary that violates this property is one that simply returns the challenge nym^* (after voting for it) in a SignRep query. The reply to such a query immediately reveals b, as expected by the semantics of the scheme ($i = i_b^*$ iff the reply is not $\perp$). Given these rules, we define receiver anonymity as follows.

Definition 3. *A scheme for signatures of reputation is* receiver anonymous *if, for all PPT adversaries $\mathcal{A}$, $|\mathbf{Pr}^{\mathsf{RANON}}_{\mathcal{A}}(\lambda) - \frac{1}{2}|$ is a negligible function of λ.*

Voter anonymity. As explained in Section 1.3, defining voter anonymity is somewhat more difficult than defining receiver anonymity. Because we require a SignRep algorithm to demonstrate the number of votes from distinct users, such an algorithm can be used by a vote receiver to determine whether two votes cast for any of their pseudonyms were produced by the same voter (duplicates). That is, the receiver can try to use the two votes to produce a signature and then check the reputation of the result with VerifyRep. Our aim in defining voter anonymity is to allow precisely this type of duplicate detection, but nothing more. The game below captures this property.

$$\mathbf{Pr}^{\mathsf{VANON}}_{\mathcal{A}}(\lambda) = \Pr\left[\begin{array}{c|c} b = b' \ \wedge \\ \textsc{Legal}(j_0^*, j_1^*, \mathsf{nym}^*, \\ \quad \mathsf{openkey}, L) & \begin{array}{c} (\mathsf{params}, \mathsf{authkey}, \mathsf{openkey}) \leftarrow \textsc{Setup}'(1^\lambda); \\ [\mathsf{cred}_i \leftarrow \textsc{GenCred}(\mathsf{params}, \mathsf{authkey})]_{1 \le i \le n}\,; \\ (j_0^*, j_1^*, \mathsf{nym}^*, \mathsf{st}) \leftarrow \mathcal{A}^{\mathcal{O}_L}(\mathsf{params});\ b \xleftarrow{R} \{0,1\}; \\ \mathsf{vt}^* \leftarrow \textsc{Vote}(\mathsf{params}, \mathsf{cred}_{j_b^*}, \mathsf{nym}^*);\ b' \leftarrow \mathcal{A}^{\mathcal{O}_L}(\mathsf{st}, \mathsf{vt}^*) \end{array} \end{array}\right]$$

In this case, we define Legal to check the following. Let $i^* = \textsc{OpenNym}(\mathsf{openkey}, \mathsf{nym}^*)$. First, if the adversary has made a query $(\text{"signrep"}, i^*, V, \mathsf{msg}) \in L$ where $\mathsf{vt}^* \in V$, we

[1] One might try to extend this definition to incorporate a forward security property ensuring pseudonyms generated before a user's credentials are compromised remain unlinkable (that is, by updating the credentials after generating each pseudonym). However, this is futile: if the updated credential can still use votes cast for old pseudonyms, then Vote and SignRep can be used to detect the old pseudonyms.

require that $\{j_0^*, j_1^*\} \subseteq \textsc{OpenVoters}(\mathsf{openkey}, V)$. In other words, the coin b should not determine the number of distinct voters in a "signrep" query involving vt^*. Second, if ("corrupt", i^*) $\in L$, we require that there not exist a ("vote", j, nym) $\in L$ such that $j \in \{j_0^*, j_1^*\}$ and $i^* = \textsc{OpenNym}(\mathsf{openkey}, \mathsf{nym})$. That is, if the adversary controls the receiver i^* of the challenge vote, then they may not request another vote from j_0^* or j_1^*, since its status as a duplicate or lack thereof would reveal b.

Definition 4. *A scheme for signatures of reputation is* voter anonymous *if, for all PPT adversaries $\mathcal{A}$, $|\mathbf{Pr}_{\mathcal{A}}^{\mathsf{VANON}}(\lambda) - \frac{1}{2}|$ is a negligible function of λ.*

Signer anonymity. The signer anonymity property requires that a signature of reputation reveal nothing about the signer beyond their reputation. In this case, we allow the adversary access to all user's credentials. As a result, they have no need for the oracle $\mathcal{O}$, as the adversary could answer the queries itself.

$$\mathbf{Pr}_{\mathcal{A}}^{\mathsf{SANON}}(\lambda) = \Pr\left[\begin{array}{c|c} b = b' \ \wedge \\ \textsc{Legal}(\mathsf{msg}, \Sigma_0^*, \Sigma_1^*) & \begin{array}{c} (\mathsf{params}, \mathsf{authkey}, \mathsf{openkey}) \leftarrow \textsc{Setup}'(1^\lambda); \\ [\mathsf{cred}_i \leftarrow \textsc{GenCred}(\mathsf{params}, \mathsf{authkey})]_{1 \le i \le n}; \\ (i_0^*, i_1^*, V_0^*, V_1^*, \mathsf{msg}, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{params}, [\mathsf{cred}_i]);\ b \xleftarrow{R} \{0,1\}; \\ \Sigma_b^* \leftarrow \textsc{SignRep}(\mathsf{params}, \mathsf{cred}_{i_b^*}, V_b^*, \mathsf{msg});\ b' \leftarrow \mathcal{A}(\mathsf{st}, \Sigma_b^*) \end{array} \end{array}\right]$$

Here, Legal requires only that $\textsc{VerifyRep}(\mathsf{params}, \mathsf{msg}, \Sigma_0^*) = \textsc{VerifyRep}(\mathsf{params}, \mathsf{msg}, \Sigma_1^*)$. That is, the value of b should affect neither the reputation values of the resulting signatures nor their validity.

Definition 5. *A scheme for signatures of reputation is* signer anonymous *if, for all PPT adversaries $\mathcal{A}$, $|\mathbf{Pr}_{\mathcal{A}}^{\mathsf{SANON}}(\lambda) - \frac{1}{2}|$ is a negligible function of λ.*

Reputation soundness. To define the soundness of a scheme for signatures of reputation, we use a computational game in which the adversary must forge a valid signature of some reputation strictly greater than that of any signature they could have produced through legitimate use of the scheme.

$$\mathbf{Pr}_{\mathcal{A}}^{\mathsf{SOUND}}(\lambda) = \Pr\left[\begin{array}{c|c} \textsc{VerifyRep}(\mathsf{params}, \mathsf{msg}, \Sigma) \neq \perp \\ \wedge\ \textsc{Legal}(\mathsf{openkey}, L, \mathsf{msg}, \Sigma) & \begin{array}{c} (\mathsf{params}, \mathsf{authkey}, \mathsf{openkey}) \leftarrow \textsc{Setup}'(1^\lambda); \\ [\mathsf{cred}_i \leftarrow \textsc{GenCred}(\mathsf{params}, \mathsf{authkey})]_{1 \le i \le n}; \\ (\mathsf{msg}, \Sigma) \leftarrow \mathcal{A}^{\mathcal{O}_L}(\mathsf{params}) \end{array} \end{array}\right]$$

In this case, Legal makes the requirement that $\Sigma \notin L$. More subtly, it must also ensure that the forged signature has reputation greater than what it could be if the adversary had used the scheme normally. The corresponding requirement checked by Legal may be formalized as follows. Let $C = \{\ i \mid (\text{"corrupt"}, i) \in L\ \}$ be the set of corrupted users. For each $i \in C$, define $S_i = \{\ j \mid (\text{"vote"}, j, \mathsf{nym}) \in L \wedge j \notin C \wedge i = \textsc{OpenNym}(\mathsf{openkey}, \mathsf{nym})\ \}$. Let $\ell_1 = |C|$, and let $\ell_2 = \max_{i \in C} |S_i|$. That is, ℓ_2 is the greatest number of distinct honest users that voted for a single corrupt user. Then we require that $\textsc{VerifyRep}(\mathsf{params}, \mathsf{msg}, \Sigma) > \ell_1 + \ell_2$ for the adversary to succeed.

Definition 6. *A scheme for signatures of reputation is* sound *if, for all PPT adversaries* $\mathcal{A}$, $\mathbf{Pr}_{\mathcal{A}}^{\mathsf{SOUND}}(\lambda)$ *is a negligible function of* λ.

In some applications, a weaker version of soundness may suffice and may be desirable for greater efficiency. One natural way to relax the definition is to specify an additional security parameter $0 \leq \varepsilon < 1$ as a multiplicative bound on the severity of cheating we wish to prevent. That is, we require a signature of reputation c to ensure that at least $(1-\varepsilon) \cdot c$ distinct votes for the signer exist. To this end, we define $\mathbf{Pr}_{\mathcal{A}}^{\varepsilon-\mathsf{SOUND}}(\lambda)$ the same way as $\mathbf{Pr}_{\mathcal{A}}^{\mathsf{SOUND}}(\lambda)$, but using the requirement that $(1-\varepsilon) \cdot \textsc{VerifyRep}(\mathsf{params}, \mathsf{msg}, \Sigma) > \ell_1 + \ell_2$. This yields the following definition of ε-soundness.

Definition 7. *A scheme for signatures of reputation is* ε-sound *if, for all PPT adversaries* $\mathcal{A}$, $\mathbf{Pr}_{\mathcal{A}}^{\varepsilon-\mathsf{SOUND}}(\lambda)$ *is a negligible function of* λ.

Note that Definition 6 is the special case of the above where $\varepsilon = 0$.

2.2 Building Blocks

We now describe the technical tools from which our scheme is constructed, including several standard cryptographic primitives, two specialized modules, and our complexity assumptions. First of all, our constructions rely on a bilinear map between groups of prime order p, which we denote $e : \mathbb{G} \times \widehat{\mathbb{G}} \to \mathbb{G}_T$. We also assume the availability of a collision-resistant hash function $H : \{0,1\}^* \to \mathbb{Z}_p$.

Non-interactive proof systems for bilinear groups. Our scheme makes extensive use of the recent Groth-Sahai non-interactive proof system [49]. Their techniques allow the construction of non-interactive witness-indistinguishable (NIWI) and non-interactive zero-knowledge (NIZK) proofs for pairing product equations, multi-scalar equations in $\mathbb{G}$ or $\widehat{\mathbb{G}}$, and quadratic equations in $\mathbb{Z}_p$. We now define the notation we will use to refer to this scheme. We write $\text{GS.}\textsc{Setup}(1^\lambda) \to (\mathsf{crs}, \mathsf{xk})$ to denote the setup algorithm, which outputs a common reference string crs and an extractor key xk. We use the notation introduced by Camenisch and Stadler [30] of the form $\Pi = \text{NIZK}\{\ x_1, \ldots, x_k\ :\ E_1 \wedge \ldots \wedge E_\ell\ \}$ to denote the construction of a zero-knowledge proof that a set of equations $E_1, \ldots, E_\ell$ is satisfiable. Here, $x_1, \ldots, x_k$ denote the secret witness variables. The NIZK consists of a commitment to each of the k witness variables, along with a constant size value for each of the ℓ equations. When variables other than the witnesses appear in the listed equations, those are public values which are not included in the proof. These values must be available for verification of the resulting proof, which we denote by $\text{GS.}\textsc{Verify}(\mathsf{crs}, \Pi, \langle a_1, \ldots, a_m \rangle)$. The arguments $a_1, \ldots, a_m$ are the public values; the relevant equations will be clear from context. Note that in the GS proof system, it is possible to produce a NIZK only when the equations being proved are "tractable" [48]. This condition holds for all the equations throughout our scheme, since none involve any elements of $\mathbb{G}_T$ except the identity.

Selective-tag weakly CCA-secure encryption. Next, we use a tag based encryption scheme [64], which we require to be selective-tag, weakly CCA-secure. For this we may employ the scheme due to Kiltz [55] based on the DLinear assumption. We denote its algorithms as follows.

CCAEnc.Setup$(1^\lambda) \to (\mathsf{pk}_{\text{cca}}, \mathsf{sk}_{\text{cca}})$: Generate a public, private key pair.

CCAEnc.Enc$(\mathsf{pk}_{\text{cca}}, \mathsf{tag}, \mathsf{msg}, (r, s)) \to C$: Encrypt a message under the given public key and tag using randomness $(r, s) \in \mathbb{Z}_p^2$.

CCAEnc.Dec$(\mathsf{sk}_{\text{cca}}, \mathsf{tag}, C) \to \mathsf{msg}$: Use the private key to decrypt a ciphertext encrypted under tag.

When we need to encrypt multiple elements $\vec{x} = (x_1, \ldots, x_k) \in \mathbb{G}^k$, we use the following shorthand: CCAEnc.Enc$(\mathsf{pk}_{\text{cca}}, \mathsf{tag}, \vec{x}, \vec{r})$, where $\vec{r} \in \mathbb{Z}_p^{2k}$.

Weakly EF-CMA secure signatures and strong one-time signatures. We will also use an SDH-based signature scheme due to Boneh and Boyen [15], which we denote BBSig. Let $g \xleftarrow{R} \mathbb{G}, s \xleftarrow{R} \mathbb{Z}_p$. In BBSig, the signing key is $\mathsf{sk}_{\text{bb}} = (g, s)$, the verification key is $\mathsf{vk}_{\text{bb}} = (g, g^s)$, a message $\mathsf{msg} \in \mathbb{Z}_p$ is signed by computing $\sigma_{\text{bb}} = g^{\frac{1}{s+\mathsf{msg}}}$, and a signature is checked by verifying that $e(\sigma_{\text{bb}}, g^s \cdot g^{\mathsf{msg}}) = e(g, g)$. This scheme is existentially unforgeable under weak chosen-message attack (weak EF-CMA security), where the adversary commits to the query messages at the beginning of the security game. The scheme is also a strong one-time signature scheme; in our construction, we use subscripts such as σ_{bb} and σ_{ots} to distinguish the cases where we use the scheme for its weak EF-CMA security from the cases where we use it to produce a strong one-time signature.

Signature scheme for certificates. To produce users' secret credentials in our construction, the registration authority will need to sign tuples of ℓ elements from $\mathbb{G}$. For this purpose we define the following signature scheme, denoted Cert.

Cert.Setup$(1^\lambda) \to (\mathsf{vk}_{\text{cert}}, \mathsf{sk}_{\text{cert}})$: Randomly select $\gamma \xleftarrow{R} \mathbb{Z}_p$, $\widehat{g}, \widehat{g}_0 \xleftarrow{R} \widehat{\mathbb{G}}$, $g, h, f_1, f_2 \xleftarrow{R} \mathbb{G}$, and, for $1 \le i \le \ell$, $\widehat{u}_i, \widehat{v}_i \xleftarrow{R} \widehat{\mathbb{G}}$. Output $\mathsf{sk}_{\text{cert}} = \gamma$ and $\mathsf{vk}_{\text{cert}} = (g, h, f_1, f_2, \widehat{g}, \widehat{g}_0, g^\gamma, \widehat{u}_1, \ldots, \widehat{u}_\ell, \widehat{v}_1, \ldots, \widehat{v}_\ell)$.

Cert.Sign$(\mathsf{vk}_{\text{cert}}, \mathsf{sk}_{\text{cert}}, \mathsf{msg}) \to \sigma$: Given an ℓ element message $\mathsf{msg} = (x_1, \ldots, x_\ell)$ in $\mathbb{G}^\ell$, select $\rho, r_1, \ldots, r_\ell, s_1, \ldots, s_\ell \xleftarrow{R} \mathbb{Z}_p$ and compute the signature as $\sigma = (\sigma_\rho, g^\rho, h^\rho, \widehat{g}_0^\rho, \{\sigma_{r_i}, \sigma_{s_i}, g^{r_i}, h^{s_i}, \widehat{u}_i^{r_i}, \widehat{v}_i^{s_i}, (x_i f_1)^{r_i}, (x_i f_2)^{s_i}\}_{1 \le i \le \ell})$, where $\sigma_\rho = \widehat{g}^{\frac{1}{\gamma+\rho}}$, $\sigma_{r_i} = \widehat{g}^{\frac{1}{\rho+r_i}}$, and $\sigma_{s_i} = \widehat{g}^{\frac{1}{\rho+s_i}}$.

CERT.VERIFY($\mathsf{vk}_{\text{cert}}, \mathsf{msg}, \sigma) \to 1$ or 0: To check a signature σ, we verify that $e(g^\gamma g^\rho,\ \sigma_\rho) = e(g, \widehat{g})$, $e(g^\rho, \widehat{g}_0) = e(g, \widehat{g}_0^\rho)$, $e(h^\rho, \widehat{g}_0) = e(h, \widehat{g}_0^\rho)$ and that, for $1 \leq i \leq \ell$, $e(g^\rho g^{r_i}, \sigma_{r_i}) = e(g, \widehat{g})$, $e(h^\rho h^{r_i}, \sigma_{s_i}) = e(h, \widehat{g})$, $e(g^{r_i}, \widehat{u}_i) = e(g, \widehat{u}_i^{r_i})$, $e(h^{s_i}, \widehat{v}_i) = e(h, \widehat{v}_i^{s_i})$, $e(x_i f_1, \widehat{u}_i^{r_i}) = e((x_i f_1)^{r_i}, \widehat{u}_i)$, and $e(x_i f_2, \widehat{v}_i^{s_i}) = e((x_i f_2)^{s_i}, \widehat{v}_i)$.

The basic idea of CERT.SIGN is to first use the long-term signing key γ to sign a one-time signing key ρ, then use ρ to sign random numbers r_i and s_i, which are in turn used to sign the components of the message. In Appendix B, we prove that this scheme (like BBSIG) satisfies weak EF-CMA security.

Key-private encryption. Our construction also makes use of an IK-CPA secure (a.k.a. key-private) encryption scheme which offers a multiplicative homomorphism. Informally, the key privacy property ensures it is infeasible to match a ciphertext with the public key used to produce it; this property is used to achieve receiver anonymity. Below, we give an IK-CPA secure scheme which may be regarded as a variant of linear encryption [16].

IKENC.SETUP$(1^\lambda) \to \mathsf{params}_{\text{ike}}$: Select $\mathsf{params}_{\text{ike}} = (f, h) \xleftarrow{R} \mathbb{G}^2$.

IKENC.GENKEY$(\mathsf{params}_{\text{ike}}) \to (\mathsf{upk}_{\text{ike}}, \mathsf{usk}_{\text{ike}})$: To generate a key pair, select $\mathsf{usk}_{\text{ike}} = (a, b) \xleftarrow{R} \mathbb{Z}_p^2$ and compute $\mathsf{upk}_{\text{ike}} = (f^a, h^b) \in \mathbb{G}^2$.

IKENC.ENC$(\mathsf{params}_{\text{ike}}, \mathsf{upk}_{\text{ike}}, \mathsf{msg}, (r, s)) \to C$: To encrypt a $\mathsf{msg} \in \mathbb{G}$ under public key $\mathsf{upk}_{\text{ike}} = (A, B)$ using random exponents $r, s \in \mathbb{Z}_p$, compute $C = (\mathsf{msg} \cdot A^r B^s, f^r, h^s)$.

IKENC.DEC$(\mathsf{params}_{\text{ike}}, \mathsf{usk}_{\text{ike}}, C) \to \mathsf{msg}$: To decrypt a ciphertext $C = (C_1, C_2, C_3)$ with private key $\mathsf{usk}_{\text{ike}} = (a, b)$, compute $\mathsf{msg} = C_1 \cdot C_2^{-a} \cdot C_3^{-b}$.

To denote encryption of a k-block message $\vec{x} \in \mathbb{G}^k$, we will use the shorthand IKENC.ENC$(\mathsf{params}_{\text{ike}}, \mathsf{upk}_{\text{ike}}, \vec{x}, \vec{r})$, where $\vec{r} \in \mathbb{Z}_p^{2k}$. In Appendix B, we provide a formal definition of IK-CPA security and a proof the above scheme meets it.

The multiplicative homomorphism of this encryption scheme may be evaluated through component-wise multiplication, denoted $\otimes$. Specifically, if (C_1, C_2, C_3) and (C_1', C_2', C_3') are encryptions of x and x' using the same $\mathsf{upk}_{\text{ike}}$ and exponents r, s and r', s' respectively, then $(C_1, C_2, C_3) \otimes (C_1', C_2', C_3')$ is the encryption of $x \cdot x'$ under $r + r'$ and $s + s'$. Also, we will write $(C_1, C_2, C_3) \odot x'$ to denote $(C_1 \cdot x', C_2, C_3)$, which is an encryption of $x \cdot x'$ under the original randomness r, s.

When using the above homomorphism to compute an encryption of $x \cdot x'$, the distribution of the resulting ciphertext is dependent on that of the input ciphertexts. In our scheme, we will need to rerandomize the ciphertexts to remove this dependency. Furthermore, we will need to do so without knowledge of the $\mathsf{upk}_{\text{ike}}$ used for encryption. Observe that this is possible if we have available two encryptions of $1 \in \mathbb{G}$ under independent random exponents. Specifically, suppose C_x is an encryption of x using r_x, s_x and C_1, C_2 are encryptions of 1

using r_1, s_1 and r_2, s_2, respectively. Select $t_1, t_2 \xleftarrow{R} \mathbb{Z}_p$ and compute $C'_x = C_x \otimes C_1^{t_1} \otimes C_2^{t_2}$, where $C_1^{t_1}$ and $C_2^{t_2}$ denote componentwise exponentiation. Then C'_x is an encryption of x using exponents $r_x + r_1 t_1 + r_2 t_2$ and $s_x + s_1 t_1 + s_2 t_2$, and the distribution of C'_x is independent of the distribution of C_x.

Assumptions. Here we detail the complexity assumptions necessary to prove the privacy and security properties of our constructions. In addition to the well-known decisional linear (DLinear) and strong Diffie-Hellman (SDH) assumptions, we employ the following three assumptions, the first two of which are parameterized by a positive integer q.

BB-HSDH Select $\gamma \xleftarrow{R} \mathbb{Z}_p^*$, $g \xleftarrow{R} \mathbb{G}$, $\widehat{g}, \widehat{g}_0 \xleftarrow{R} \widehat{\mathbb{G}}$, and $\rho_i \xleftarrow{R} \mathbb{Z}_p$ for $i \in \{1, \ldots, q\}$. Then given $(g, g^\gamma, \widehat{g}, \widehat{g}^\gamma, \widehat{g}_0, (\rho_i, \widehat{g}^{\frac{1}{\gamma+\rho_i}})_{1 \le i \le q})$, it is computationally infeasible to output a tuple $(g^\rho, \widehat{g}_0^\rho, \widehat{g}^{\frac{1}{\gamma+\rho}})$ where $\rho \notin \{\rho_1, \ldots, \rho_q\}$.

BB-CDH Select $\gamma \xleftarrow{R} \mathbb{Z}_p^*$, $g \xleftarrow{R} \mathbb{G}$, $\widehat{g}, \widehat{u} \xleftarrow{R} \widehat{\mathbb{G}}$, and $\rho_i \xleftarrow{R} \mathbb{Z}_p$ for $i \in \{1, \ldots, q\}$. Then given $(g, g^\gamma, \widehat{g}, \widehat{g}^\gamma, \widehat{u}, (\rho_i, \widehat{g}^{\frac{1}{\gamma+\rho_i}})_{1 \le i \le q})$, it is infeasible to output $\widehat{u}^\gamma$.

SCDH Select $\rho, r, s \xleftarrow{R} \mathbb{Z}_p$, $g, h \xleftarrow{R} \mathbb{G}$, and $\widehat{g}, \widehat{u}, \widehat{v} \xleftarrow{R} \widehat{\mathbb{G}}$. Then given $(\rho, g, h, \widehat{g}, \widehat{u}, \widehat{v}, \widehat{u}^r, \widehat{v}^s, g^r, h^s, \widehat{g}^{\frac{1}{r+\rho}}, \widehat{g}^{\frac{1}{s+\rho}})$ it is infeasible to output a tuple (z, z^r, z^s) where $z \in \mathbb{G}$ and $z \neq 1$.

The first two assumptions above were introduced in the delegatable anonymous credential work of Belenkiy et. al. [8]. The SCDH ("stronger than CDH") assumption is new; we provide a proof of its hardness in generic groups in Appendix A. Note that if we remove the terms $\widehat{g}^{\frac{1}{r+\rho}}$ and $\widehat{g}^{\frac{1}{s+\rho}}$ from the SCDH assumption, the resulting assumption would be implied by DLinear. Therefore, we are assuming that these two terms will not help the adversary in outputting (z, z^r, z^s).

2.3 Algorithms

To better motivate our full construction, we first present a simpler, "unblinded" version which neglects the receiver anonymity property and assumes users correctly follow the protocol. The algorithms of this unblinded scheme will form part of the full version.

Unblinded scheme. In unblinded scheme, each user i generates a voting key $\mathsf{votekey}_i$ and a receiver key rcvkey_i. Given rcvkey_i, a user j can use its $\mathsf{votekey}_j$ to compute an unblinded vote $U_{j,i}$ for user i. User i can then demonstrate its reputation by showing a "weak encryption" of the unblinded votes it has received.

$\textsc{SetupUnblinded}(1^\lambda) \to \mathsf{params}_{\mathrm{ub}}$: Select $\mathsf{params}_{\mathrm{ub}} = (g, h) \xleftarrow{R} \mathbb{G}^2$.

GenVoteKey($\mathsf{params}_{\mathrm{ub}}$) $\to \mathsf{rcvkey}_i, \mathsf{votekey}_i$: To make a key pair for user i, select $\alpha_i, \beta_i \xleftarrow{R} \mathbb{Z}_p$ and $x_{i,k}, y_{i,k}, z_{i,k} \xleftarrow{R} \mathbb{G}$ for $k \in \{1, 2\}$. Define $\mathsf{vsk}_i = (\alpha_i, \beta_i)$ and $\mathsf{vpk}_i = (g^{\alpha_i}, h^{\beta_i}, z_{i,1}, z_{i,2})$ and output $\mathsf{rcvkey}_i = (x_{i,1}, y_{i,1}, x_{i,2}, y_{i,2})$ and $\mathsf{votekey}_i = (\mathsf{vsk}_i, \mathsf{vpk}_i)$.

VoteUnblinded($\mathsf{rcvkey}_i, \mathsf{votekey}_j$) $\to U_{j,i}$: To compute a vote from j to i, we parse the keys as above, using subscripts i, j to distinguish the components of user i's key and user j's key, then output $U_{j,i} = (x_{i,1}^{\alpha_j} \cdot y_{i,1}^{\beta_j} \cdot z_{j,1},\ x_{i,2}^{\alpha_j} \cdot y_{i,2}^{\beta_j} \cdot z_{j,2})$.

ShowRep($U_1, \ldots, U_c, (r, s)$) $\to \mathsf{rep}$: To compute the weakly encrypted version of c unblinded votes using random exponents $r, s \in \mathbb{Z}_p$, we output $\mathsf{rep} = [\ u_{i,1}^r \cdot u_{i,2}^s\ ]_{1 \le i \le c}$, where $u_{i,1}, u_{i,2}$ denote the two components of U_i.

These algorithms are designed to ensure several properties we will need when they are used within the full construction. First, an unblinded vote $U_{j,i}$ is a deterministic function of $\mathsf{votekey}_j$ and rcvkey_i, so votes $U_{j_1,i}$, $U_{j_2,i}$ from distinct voters $j_1 \neq j_2$ will have distinct values. Furthermore, ShowRep preserves this distinctness, so if $U_{j_1,i} \neq U_{j_2,i}$ and $(V_{j_1,i}, V_{j_2,i}) = \text{ShowRep}(U_{j_1,i}, U_{j_2,i}, (r, s))$, then $V_{j_1,i} \neq V_{j_2,i}$. Second, without $\mathsf{votekey}_j$, an adversary cannot forge a vote from user j (based on the CDH assumption). These two properties will be needed for the soundness of the full construction. Third, given U_{j_1,i_1}, U_{j_2,i_2}, two colluding receivers i_1 and i_2 cannot determine whether $j_1 = j_2$.[2] This relies on the DLinear assumption and will help ensure voter anonymity. Finally, if ShowRep is invoked twice on the same unblinded votes but with independent randomness, the resulting values rep_1 and rep_2 cannot be linked to one another. This also relies on the DLinear assumption and will be used to help ensure signer anonymity.

Full construction. The algorithms of our full construction are given at the end of this chapter in Figures 2.1 through 2.4. As for the opening algorithms, Setup′ is obtained from Setup by simply returning the extractor key xk of the Groth-Sahai proof system as $\mathsf{openkey} = \mathsf{xk}$ rather than discarding it. The OpenNym algorithm then uses the extractor key on the NIZK in a nym to obtain the committed rcvkey, which may then be matched against a list of credentials $\mathsf{cred}_1, \ldots, \mathsf{cred}_n$ to determine the owner of nym. Similarly, OpenVote works by using the extractor key to obtain the rcvkey of the voter from the commitment in the vote's NIZK.

The algorithms of Figures 2.1 through 2.4 (along with opening algorithms described above) satisfy Definitions 1–6. The correctness properties may be verified by inspection; for each of the other properties, we provide proofs in Appendix B. Intuitively, the full scheme is obtained through three modifications to the unblinded scheme. First, to limit voting to valid members of the system, a user's rcvkey and $\mathsf{votekey}$ are signed and issued by the registration

[2]Note that, if the term $z_{j,k}$ is omitted, an attack is possible. Two colluding users vote for a recipient i, resulting in two votes U_1, U_2. Later, when i constructs $\mathsf{rep} = \text{ShowRep}(U_1, U_2, (r, s))$, the adversary would be able to detect the correlation in rep and confirm that it came from i. The term $z_{j,k}$ prevents this because the adversary does not know its exponent.

authority. Second, we use a "blinded" voting protocol based on key-private, homomorphic encryption to achieve receiver anonymity. Third, users construct NIZKs to prove they have correctly followed the protocol. We now elaborate on the later two ideas and the operation of the SignRep algorithm.

Blinded voting. From a high level, a user computes a one-time pseudonym nym by encrypting their rcvkey under their $\mathsf{upk}_{\text{ike}}$. Instead of voting on the rcvkey, a voter then votes on the encrypted version in the nym. This is made possible by the homomorphism of the encryption scheme: the voter homomorphically computes an encryption of the unblinded vote, which the recipient can later decrypt. Only the recipient has the secret key $\mathsf{usk}_{\text{ike}}$ necessary to do so.

More precisely, if $(C_{x,k}, C_{y,k})_{k\in\{1,2\}}$ is the encryption of $\mathsf{rcvkey} = (x_1, y_1, x_2, y_2)$, the voter computes the encrypted vote as $(C_{x,k}^{\alpha} \otimes C_{y,k}^{\beta} \odot z_k)$, where α, β, z_k come from the voter's key. To allow the voter to rerandomize the resulting ciphertext using the technique described in Section 2.2, the recipient also includes two independent encryptions of $1 \in \mathbb{G}$ in the nym, which we denote $C_{1,1}$ and $C_{1,2}$. To understand the requirement that IKEnc be selective-tag weakly CCA-secure, recall that in the security definition, when an adversary makes a "signrep" oracle query, it can indirectly learn whether one or more votes correspond to the user i. This allows the oracle to be used as something similar to a decryption oracle, ultimately requiring a CCA security property. To ensure an adversary cannot frame a honest user i by forging a nym that opens to user i, the GenNym algorithm also picks a one-time signature key pair $(\mathsf{sk}_{\text{ots}}, \mathsf{vk}_{\text{ots}})$ and proves knowledge of a signature $\sigma_{\text{bb}} \leftarrow \text{BBSig.Sign}(\mathsf{sk}_{\text{bb}}, H(\mathsf{vk}_{\text{ots}}))$, then uses sk_{ots} to sign the entire nym. One-time signatures are similarly employed in Groth's group signature scheme [48]; we also use this technique in the votes and signatures of reputation.

Nested NIZKs. Users must prove through a series of NIZKs that they have correctly followed the algorithms using credentials certified by the registration authority. It is worth mentioning that we use "nested" NIZKs. Specifically, a signature of reputation includes a commitment to the votes and a NIZK proving they are valid. Because the votes themselves contain NIZKs, proving that the votes are valid involves proving that the NIZKs they contain satisfy the Groth-Sahai NIZK verification equations, all within the NIZK for the resulting signature.

Signatures of reputation. To construct a signature of reputation, the signer uses $\mathsf{usk}_{\text{ike}}$ to decrypt the ciphertexts in the votes they have received, obtaining unblinded votes $U_1, \ldots, U_c$. It calls $\text{ShowRep}(U_1, \ldots, U_c)$ to compute a weak encryption of these unblinded votes. Recall that this encryption preserves "distinctness." It also encrypts these unblinded votes using CCAEnc; this allows a simulator to open the signature of reputation under a simulated crs without xk.

2.4 Short Signatures of Reputation

What we have described thus far produces signatures of reputation c that are of size $\Theta(c)$. If perfect soundness is not necessary, this cost can be dramatically reduced. Specifically, in this section we describe a way to obtain signatures of size $O(\frac{1}{\varepsilon}\log c)$ while maintaining ε-soundness (Definition 7) in the random oracle model.

From a high level, we take the following approach in improving space efficiency. Rather than including all votes in the signature, we only include a randomly selected, constant size subset. Specifically, we require that the signer first commit to a list of all the votes with a hash function H and then interpret the output of H as a challenge specifying the indices of the votes to include. The signer must also demonstrate that the votes included were in fact the votes at the indices in the challenge set when the commitment was formed. To do so efficiently, we compute the commitment using a Merkle hash tree [67]. We implement this technique with the following changes to the SignRep algorithm.

After determining the number of distinct unblinded votes c, set $\ell = \lceil \frac{\lambda}{\varepsilon} \rceil$; this will be the size of our challenge set. Recall that λ is the security parameter. Now if $\ell \geq c$, we include all votes, computing Σ normally. Otherwise, we proceed as follows. Let $\mathsf{rep} = (R_1, \ldots R_c)$. Sort these values to obtain a list $R_{\rho_1}, R_{\rho_2}, \ldots, R_{\rho_c}$, where $R_{\rho_i} < R_{\rho_{i+1}}$ for $1 \leq i \leq c-1$. The NIZK Π computed by SignRep will include the following values for the ρ_ith vote: a tuple θ_i of commitments to $\mathsf{vt}_i, \mathsf{tag}_i, U_i$ and a tuple of values ζ_i used to verify the GS.Verify and IKEnc.Dec equations. We collect these together to form $\omega_i = (i, R_{\rho_i}, R_{\rho_{i+1}}, \theta_i, \zeta_i)$, with $R_{\rho_{c+1}}$ defined as a special symbol ∞ for consistency.

Now we can compute the set of challenge indices. Let $m = \lceil \log_2 c \rceil$ be the height of the smallest binary tree with at least c leaf nodes. We construct a complete binary tree of height m, associating the first c leaf nodes with the values $\omega_1, \omega_2, \ldots, \omega_c$ and any remaining leaf nodes with dummy values $\omega_{c+1}, \ldots, \omega_{2^m} = 0$. Next, we compute a hash value h_n for each node n in the tree as follows. If n is a leaf with index i, we set $h_n = H(\omega_i)$; otherwise, n has a left child n_l and a right child n_r and we set $h_n = H(h_{n_l} \| h_{n_r})$. We thus obtain a hash value h_{root} for the root of the tree to be used to construct a set of ℓ distinct challenge indices $I \subset \{1, 2, \ldots, c\}$. This is done by starting with an empty set I, and adding the indices $1 + (H(0\|h_{\text{root}}) \bmod c)$, $1 + (H(1\|h_{\text{root}}) \bmod c)$, etc., one by one, skipping duplicates and stopping when $|I| = \ell$.

Now that we have specified the challenge set I, we may list the values included in the final signature of reputation. We start with the proof Π computed as before and remove all per-vote values $\theta_{\rho_i}, \zeta_{\rho_i}$ for $i \notin I$. Note that the result is a valid Groth-Sahai NIZK that only verifies the votes at indices in I; furthermore, it is distributed identically to a proof computed directly using only those votes. In addition to the reduced proof Π, we include in the final signature of reputation the pairs $(R_{\rho_i}, R_{\rho_{i+1}})$ for each $i \in I$ and the off-path hashes needed to verify that the challenge set was constructed correctly. There are precisely $\lceil \log_2 c \rceil$ off-path hash values for each vote (although some will be shared by multiple votes), so we obtain an overall signature size of $O(\ell \log c) = O(\frac{1}{\varepsilon}\log c)$.

The necessary modifications to the VerifyRep algorithm are straightforward. We verify the proof Π normally, then collect each of the per-vote terms present and hash them with H to obtain the values of the corresponding leaves in the hash tree. Using the provided off-path hash values, we recompute the root value h_{root}. From h_{root}, we compute the challenge set I, and then we check that it corresponds to the votes that were included. Also, for each pair $(R_{\rho_i}, R_{\rho_{i+1}})$, we check that $R_{\rho_i} < R_{\rho_{i+1}}$.

Let SignRep$'$ and VerifyRep$'$ denote the modified versions of the SignRep and VerifyRep algorithms as described above. Then the algorithms Setup, GenCred, GenNym, Vote, SignRep$'$, and VerifyRep$'$ constitute an ε-sound scheme for signatures of reputation according to Definition 7. In Appendix B, we prove this in the random oracle model.

$\underline{\textsc{Setup}}(1^\lambda)$
$(\mathsf{crs}, \mathsf{xk}) \leftarrow \text{GS.}\textsc{Setup}(1^\lambda)$
$\mathsf{params}_{\mathrm{ub}} \leftarrow \textsc{SetupUnblinded}(1^\lambda)$
$\mathsf{params}_{\mathrm{ike}} \leftarrow \text{IKE}\textsc{nc.Setup}(1^\lambda)$
$(\mathsf{pk}_{\mathrm{cca}}, \mathsf{sk}_{\mathrm{cca}}) \leftarrow \text{CCAE}\textsc{nc.Setup}(1^\lambda)$
$(\mathsf{vk}_{\mathrm{cert}}, \mathsf{sk}_{\mathrm{cert}}) \leftarrow \textsc{Cert.Setup}(1^\lambda)$
Return $\mathsf{params} = (\mathsf{crs}, \mathsf{params}_{\mathrm{ub}}, \mathsf{params}_{\mathrm{ike}}, \mathsf{pk}_{\mathrm{cca}}, \mathsf{vk}_{\mathrm{cert}}),\ \ \mathsf{authkey} = \mathsf{sk}_{\mathrm{cert}}$

$\underline{\textsc{GenCred}}(\mathsf{params}, \mathsf{authkey})$
$(\mathsf{rcvkey}, \mathsf{vsk}, \mathsf{vpk}) \leftarrow \textsc{GenVoteKey}(\mathsf{params}_{\mathrm{ub}})$
$(\mathsf{vk}_{\mathrm{bb}}, \mathsf{sk}_{\mathrm{bb}}) \leftarrow \text{BBS}\textsc{ig.Setup}(1^\lambda)$
$(\mathsf{upk}_{\mathrm{ike}}, \mathsf{usk}_{\mathrm{ike}}) \leftarrow \text{IKE}\textsc{nc.GenKey}(\mathsf{params}_{\mathrm{ike}})$
$\mathsf{cert} \leftarrow \textsc{Cert.Sign}(\mathsf{vk}_{\mathrm{cert}}, \mathsf{sk}_{\mathrm{cert}}, \langle \mathsf{rcvkey}, \mathsf{vpk}, \mathsf{vk}_{\mathrm{bb}}, \mathsf{upk}_{\mathrm{ike}} \rangle)$
Return $\mathsf{cred} = (\mathsf{rcvkey}, \mathsf{vpk}, \mathsf{vk}_{\mathrm{bb}}, \mathsf{upk}_{\mathrm{ike}}, \mathsf{cert}, \mathsf{sk}_{\mathrm{bb}}, \mathsf{vsk}, \mathsf{usk}_{\mathrm{ike}})$

$\underline{\textsc{GenNym}}(\mathsf{params}, \mathsf{cred})$
Parse $\mathsf{cred} = (\mathsf{rcvkey}, \mathsf{vpk}, \mathsf{vk}_{\mathrm{bb}}, \mathsf{upk}_{\mathrm{ike}}, \mathsf{cert}, \mathsf{sk}_{\mathrm{bb}}, \mathsf{vsk}, \mathsf{usk}_{\mathrm{ike}})$
Denote $\mathsf{msg} = (\mathsf{rcvkey}, 1, 1) \in \mathbb{G}^6$
$(\mathsf{vk}_{\mathrm{ots}}, \mathsf{sk}_{\mathrm{ots}}) \leftarrow \text{BBS}\textsc{ig.Setup}(1^\lambda),\ \ \vec{r} \xleftarrow{R} \mathbb{Z}_p^{12}$
$C \leftarrow \text{IKE}\textsc{nc.Enc}(\mathsf{params}_{\mathrm{ike}}, \mathsf{upk}_{\mathrm{ike}}, \mathsf{msg}, \vec{r})$
$\sigma_{\mathrm{bb}} \leftarrow \text{BBS}\textsc{ig.Sign}(\mathsf{sk}_{\mathrm{bb}}, H(\mathsf{vk}_{\mathrm{ots}}))$
$\Pi = \text{NIZK}\{\ \mathsf{rcvkey}, \mathsf{vpk}, \mathsf{vk}_{\mathrm{bb}}, \mathsf{upk}_{\mathrm{ike}}, \mathsf{cert}, \sigma_{\mathrm{bb}}, \vec{r}\ :$
$\quad \textsc{Cert.Verify}(\mathsf{vk}_{\mathrm{cert}}, \langle \mathsf{rcvkey}, \mathsf{vpk}, \mathsf{vk}_{\mathrm{bb}}, \mathsf{upk}_{\mathrm{ike}} \rangle, \mathsf{cert})$
$\quad \wedge\ \text{BBS}\textsc{ig.Verify}(\mathsf{vk}_{\mathrm{bb}}, H(\mathsf{vk}_{\mathrm{ots}}), \sigma_{\mathrm{bb}})$
$\quad \wedge\ C = \text{IKE}\textsc{nc.Enc}(\mathsf{params}_{\mathrm{ike}}, \mathsf{upk}_{\mathrm{ike}}, \mathsf{msg}, \vec{r})\}$
$\sigma_{\mathrm{ots}} \leftarrow \text{BBS}\textsc{ig.Sign}(\mathsf{sk}_{\mathrm{ots}}, H(C \| \Pi \| \mathsf{vk}_{\mathrm{ots}}))$
Return $\mathsf{nym} = (C, \Pi, \mathsf{vk}_{\mathrm{ots}}, \sigma_{\mathrm{ots}})$

Figure 2.1: The $\textsc{Setup}$, $\textsc{GenCred}$, and $\textsc{GenNym}$ algorithms.

$\underline{\text{VOTE}}(\mathsf{params}, \mathsf{cred}, \mathsf{nym})$
Parse $\mathsf{cred} = (\mathsf{rcvkey}, \mathsf{vpk}, \mathsf{vk}_{\text{bb}}, \mathsf{upk}_{\text{ike}}, \mathsf{cert}, \mathsf{sk}_{\text{bb}}, \mathsf{vsk}, \mathsf{usk}_{\text{ike}})$
Parse $\mathsf{nym} = (C, \ldots)$ where $C = (\{C_{x,k}, C_{y,k}\}_{k\in\{1,2\}}, C_{1,1}, C_{1,2})$
Parse $\mathsf{vsk} = (\alpha, \beta)$, $\mathsf{vpk} = (A, B, z_1, z_2)$
Parse $\mathsf{rcvkey} = (x_1, \ldots)$
If $\neg$VERIFYNYM$(\mathsf{params}, \mathsf{nym})$ Return $\perp$
$(\mathsf{vk}_{\text{ots}}, \mathsf{sk}_{\text{ots}}) \leftarrow$ BBSIG.SETUP(1^λ),
$\mathsf{tag} = H(\mathsf{vk}_{\text{ots}})$, $\sigma_{\text{bb}} \leftarrow$ BBSIG.SIGN$(\mathsf{sk}_{\text{bb}}, \mathsf{tag})$
$\vec{r} = (r_{1,1}, r_{1,2}, r_{2,1}, r_{2,2}) \xleftarrow{R} \mathbb{Z}_p^4$, $\vec{s} \xleftarrow{R} \mathbb{Z}_p^2$
$C_1 = \left[(C_{x,k}^{\alpha} \otimes C_{y,k}^{\beta} \odot z_k) \otimes C_{1,1}^{r_{k,1}} \otimes C_{1,2}^{r_{k,2}}\right]_{k\in\{1,2\}}$
$C_2 \leftarrow$ CCAENC.ENC$(\mathsf{pk}_{\text{cca}}, \mathsf{tag}, x_1, \vec{s})$
$\Pi =$ NIZK$\{$ $\mathsf{rcvkey}, \mathsf{vpk}, \mathsf{vk}_{\text{bb}}, \mathsf{upk}_{\text{ike}}, \mathsf{cert}, \mathsf{vsk}, \sigma_{\text{bb}}, \vec{r}, \vec{s}$:
CERT.VERIFY$(\mathsf{vk}_{\text{cert}}, \langle \mathsf{rcvkey}, \mathsf{vpk}, \mathsf{vk}_{\text{bb}}, \mathsf{upk}_{\text{ike}} \rangle, \mathsf{cert})$
$\wedge$ BBSIG.VERIFY$(\mathsf{vk}_{\text{bb}}, \mathsf{tag}, \sigma_{\text{bb}})$
$\wedge\ A = g^{\alpha}\ \wedge\ B = h^{\beta}$
$\wedge\ C_1 = \left[(C_{x,k}^{\alpha} \otimes C_{y,k}^{\beta} \odot z_k) \otimes C_{1,1}^{r_{k,1}} \otimes C_{1,2}^{r_{k,2}}\right]_{k\in\{1,2\}}$
$\wedge\ C_2 =$ CCAENC.ENC$(\mathsf{pk}_{\text{cca}}, \mathsf{tag}, x_1, \vec{s})$ $\}$
$\sigma_{\text{ots}} \leftarrow$ BBSIG.SIGN$(\mathsf{sk}_{\text{ots}}, H(\mathsf{nym} \| C_1 \| C_2 \| \Pi \| \mathsf{vk}_{\text{ots}}))$
Return $\mathsf{vt} = (\mathsf{nym}, C_1, C_2, \Pi, \mathsf{vk}_{\text{ots}}, \sigma_{\text{ots}})$

Subroutine: $\underline{\text{VERIFYNYM}}(\mathsf{params}, \mathsf{nym})$
Parse $\mathsf{nym} = (C, \Pi, \mathsf{vk}_{\text{ots}}, \sigma_{\text{ots}})$
If BBSIG.VERIFY$(\mathsf{vk}_{\text{ots}}, H(C \| \Pi \| \mathsf{vk}_{\text{ots}}), \sigma_{\text{ots}})$
$\wedge$ GS.VERIFY$(\mathsf{crs}, \Pi, \langle \mathsf{params}, C, \mathsf{vk}_{\text{ots}} \rangle)$
Return 1; Else return 0

Figure 2.2: The VOTE algorithm.

$\underline{\text{SIGNREP}}(\mathsf{params}, \mathsf{cred}, V, \mathsf{msg})$
Parse $\mathsf{cred} = (\mathsf{rcvkey}, \mathsf{vpk}, \mathsf{vk}_{\text{bb}}, \mathsf{upk}_{\text{ike}}, \mathsf{cert}, \mathsf{sk}_{\text{bb}}, \mathsf{vsk}, \mathsf{usk}_{\text{ike}})$
Parse $\mathsf{params}_{\text{ike}} = (f, h), \quad \mathsf{upk}_{\text{ike}} = (A, B), \quad \mathsf{usk}_{\text{ike}} = (a, b)$
Parse $V = \{\mathsf{vt}_1, \mathsf{vt}_2, \ldots, \mathsf{vt}_{c'}\}$ where $\mathsf{vt}_i = (\mathsf{nym}_i, C_{1,i}, C_{2,i}, \Pi_i, \mathsf{vk}_{\text{ots},i}, \sigma_{\text{ots},i})$
Denote $\mathsf{tag}_i = H(\mathsf{vk}_{\text{ots},i})$
$\forall 1 \le i \le c' :$ Parse $\mathsf{nym}_i = (C'_i, \Pi'_i, \mathsf{vk}_{\text{ots},i}', \sigma_{\text{ots},i}')$
If $\exists 1 \le i \le c' : \ 1 \ne \text{VERIFYVOTE}(\mathsf{params}, \mathsf{vt}_i)$, return $\perp$
If $\exists 1 \le i \le c' : \mathsf{rcvkey} \ne \text{IKENC.DEC}(\mathsf{params}_{\text{ike}}, \mathsf{usk}_{\text{ike}}, C'_i)$, return $\perp$
$(\mathsf{vk}_{\text{ots}}, \mathsf{sk}_{\text{ots}}) \leftarrow \text{BBSIG.SETUP}(1^\lambda), \quad \mathsf{tag} = H(\mathsf{vk}_{\text{ots}})$
$\sigma_{\text{bb}} \leftarrow \text{BBSIG.SIGN}(\mathsf{sk}_{\text{bb}}, H(\mathsf{vk}_{\text{ots}}))$
$\forall 1 \le i \le c' : U'_i \leftarrow \text{IKENC.DEC}(\mathsf{params}_{\text{ike}}, \mathsf{usk}_{\text{ike}}, C_{1,i})$
Remove duplicates: $\{U_1, U_2, \ldots, U_c\} = \{U'_1, \ldots, U'_{c'}\}$,
where $c \le c'$ and $U_1, U_2, \ldots, U_c$ are all distinct
$\vec{r} \xleftarrow{R} \mathbb{Z}_p^2, \quad \mathsf{rep} \leftarrow \text{SHOWREP}((U_1, \ldots, U_c), \vec{r})$
$\vec{s} \xleftarrow{R} \mathbb{Z}_p^{2c}, \quad C \leftarrow \text{CCAENC.ENC}(\mathsf{pk}_{\text{cca}}, \mathsf{tag}, (U_1, \ldots, U_c), \vec{s})$
$\Pi = \text{NIZK}\{ \ \mathsf{rcvkey}, \mathsf{vpk}, \mathsf{vk}_{\text{bb}}, \mathsf{upk}_{\text{ike}}, \mathsf{cert}, \mathsf{usk}_{\text{ike}}, \sigma_{\text{bb}}, (\mathsf{vt}_i, \mathsf{tag}_i, U_i)_{1 \le i \le c}, \vec{r}, \vec{s} \ :$
$\text{CERT.VERIFY}(\mathsf{vk}_{\text{cert}}, \langle \mathsf{rcvkey}, \mathsf{vpk}, \mathsf{vk}_{\text{bb}}, \mathsf{upk}_{\text{ike}} \rangle, \mathsf{cert})$
$\wedge \ \text{BBSIG.VERIFY}(\mathsf{vk}_{\text{bb}}, \mathsf{tag}, \sigma_{\text{bb}}) \ \wedge \ A = f^a \quad = h^b$
$\wedge \ \forall 1 \le i \le c : \ \text{GS.VERIFY}(\mathsf{crs}, \Pi_i, \langle \mathsf{params}, C_{1,i}, C_{2,i}, \mathsf{tag}_i \rangle)$
$\wedge \ \forall 1 \le i \le c : \ \text{GS.VERIFY}(\mathsf{crs}, \Pi'_i, \langle \mathsf{params}, C'_i \rangle)$ (*)
$\wedge \ \forall 1 \le i \le c : \ \mathsf{rcvkey} = \text{IKENC.DEC}(\mathsf{params}_{\text{ike}}, \mathsf{usk}_{\text{ike}}, C'_i)$
$\wedge \ \forall 1 \le i \le c : \ U_i = \text{IKENC.DEC}(\mathsf{params}_{\text{ike}}, \mathsf{usk}_{\text{ike}}, C_{1,i})$
$\wedge \ \mathsf{rep} = \text{SHOWREP}((U_1, \ldots, U_c), \vec{r})$
$\wedge \ C = \text{CCAENC.ENC}(\mathsf{pk}_{\text{cca}}, \mathsf{tag}, (U_1, \ldots, U_c), \vec{s}) \ \}$
$\sigma_{\text{ots}} \leftarrow \text{BBSIG.SIGN}(\mathsf{sk}_{\text{ots}}, H(c \| \mathsf{msg} \| C \| \mathsf{rep} \| \Pi \| \mathsf{vk}_{\text{ots}}))$
Return $\Sigma = (c, \mathsf{msg}, C, \mathsf{rep}, \Pi, \mathsf{vk}_{\text{ots}}, \sigma_{\text{ots}})$
(*): Here, verify all equations in the NIZK Π'_i, except the BBSIG.VERIFY equation.

Subroutine: $\underline{\text{VERIFYVOTE}}(\mathsf{params}, \mathsf{vt})$
Parse $\mathsf{vt} = (\mathsf{nym}, C_1, C_2, \Pi, \mathsf{vk}_{\text{ots}}, \sigma_{\text{ots}})$
If $\text{BBSIG.VERIFY}(\mathsf{vk}_{\text{ots}}, H(\mathsf{nym} \| C_1 \| C_2 \| \Pi \| \mathsf{vk}_{\text{ots}}), \sigma_{\text{ots}})$
$\wedge \ \text{GS.VERIFY}(\mathsf{crs}, \Pi, \langle \mathsf{params}, C_1, C_2, \mathsf{vk}_{\text{ots}} \rangle)$
$\wedge \ \text{VERIFYNYM}(\mathsf{params}, \mathsf{nym})$
Return 1; Else return 0

Figure 2.3: The SIGNREP algorithm.

$\underline{\text{VERIFYREP}}(\mathsf{params}, \mathsf{msg}, \Sigma)$
Parse $\Sigma = (c, \mathsf{msg}, C, \mathsf{rep}, \Pi, \mathsf{vk}_{\text{ots}}, \sigma_{\text{ots}})$
If there are no duplicate values in rep
$\wedge\ |\mathsf{rep}| = c$
$\wedge\ \text{GS.VERIFY}(\mathsf{crs}, \Pi, \langle \mathsf{params}, C, \mathsf{rep}, \mathsf{vk}_{\text{ots}} \rangle)$
$\wedge\ \text{BBSIG.VERIFY}(\mathsf{vk}_{\text{ots}}, H(c \| \mathsf{msg} \| C \| \mathsf{rep} \| \Pi \| \mathsf{vk}_{\text{ots}}), \sigma_{\text{ots}})$
Return c; Else return $\perp$

Figure 2.4: The VERIFYREP algorithm.

Chapter 3

Efficient Private Stream Searching

A variety of types of information sources are made available by the Internet. These include conventional websites, time sensitive web pages such as news articles and blog posts, auctions, forums, and classified ads. One common link between all of these sources is that searching mechanisms are vital for a user to be able to distill the information relevant to him. Most search mechanisms involve a client sending a set of search criteria (e.g., a textual keyword) to a server and the server performing the search over some large data set. However, for some applications a client would like to hide their search criteria, i.e., which data they are interested in. A client might want to protect the privacy of their search queries for a variety of reasons ranging from personal privacy to protection of commercial interests. A naive method for allowing private searches would be to download the entire resource to the client machine and perform the search locally. However, this is typically infeasible due to the large size of the data set to be searched, the limited bandwidth between the client and the remote host, or to the unwillingness of the other party to disclose the entire resource to the client.

In this chapter we detail an efficient cryptographic system which would allow a wide variety of applications to conduct searches on untrusted servers while provably maintaining the secrecy of the search criteria. We begin by reviewing several tools that will be needed in our construction: Paillier's cryptosystem, the definition of a pseudo-random function family, and Bloom filters. After explaining the operation of our proposed algorithms, we will discuss their asymptotic efficiency and several extensions which provide additional features.

3.1 Preliminaries

The Paillier cryptosystem is a probabilistic, public key cryptosystem which provides semantic security under the decisional composite residuosity assumption (DCRA) [74]. As in RSA, the public key N is the product of two large primes, and its factorization is the private key. In the following discussion, the encryption of a plaintext m will be denoted $E(m)$,

and the decryption of a ciphertext c will be denoted $D(c)$. Plaintexts are represented by elements of the group $\mathbb{Z}_N$ while ciphertexts exist within $\mathbb{Z}_{N^2}$. Thus $E : \mathbb{Z}_N \to \mathbb{Z}_{N^2}^*$ and $D : \mathbb{Z}_{N^2}^* \to \mathbb{Z}_N$. Note that ciphertexts are twice as large as plaintexts.[1] The key property of the Paillier cryptosystem upon which the entire system is based is its homomorphism: for any $a, b \in \mathbb{Z}_N$, it is the case that $D(E(a) \cdot E(b)) = a + b$. That is, multiplying ciphertexts has the effect of adding the corresponding plaintexts. This allows one to perform rudimentary computations on encrypted values. Our construction may be adapted to use any public key, homomorphic cryptosystem, but for concreteness, we assume the use of the Paillier cryptosystem throughout the rest of the paper.

In our construction we also use a pseudo-random function family $G : \mathcal{K}_G \times \mathbb{Z} \times \mathbb{Z} \to \{0, 1\}$. That is, given a key k, G should map each pair of integers to a pseudo-random bit. The security of such a function family G is defined by the following game between a challenger and an adversary $\mathcal{A}$. A challenger chooses a random key $k \xleftarrow{R} \mathcal{K}_G$ and sets $g = G_k$, then flips a coin $\beta \in \{0, 1\}$. At this point the adversary submits a series of queries from the domain $\mathbb{Z} \times \mathbb{Z}$ to the challenger. If $\beta = 0$ the challenger will respond by evaluating the function g on the input, whereas if $\beta = 1$ it will respond with a random bit to all new queries, while giving the same response if the same query is made twice. Finally, the adversary outputs a guess β'. We define the adversary's advantage in this game as $\mathsf{Adv}_{\mathcal{A}} = |\mathrm{P}(\beta = \beta') - \frac{1}{2}|$. We say that a pseudo random function is $(\omega_t, \omega_q, \varepsilon)$-secure if no ω_t time adversary that makes at most ω_q oracle queries has advantage greater than ε. Interestingly, the security of the pseudo-random function family employed in our scheme is actually only necessary to prove correctness properties. Privacy is unaffected, as explained in Section 3.3. For the purpose of our constructions, we may simply select a random $k \xleftarrow{R} \mathcal{K}_G$ and provide $g = G_k$ ahead of time as a global, public parameter.

A Bloom filter [13] is a space-efficient data structure for storing a set of keys that has several unique features. First, rather than allowing direct enumeration of the keys stored, a Bloom filter only supports querying to determine if a given key is present. Second, while queries for a key that has been previously stored will always succeed, a query for a key which has *not* been previously stored will also succeed with some small, configurable probability. This false positive inducing "lossiness" allows Bloom filters to achieve extremely compact storage. A Bloom filter may be implemented as a vector of ℓ bits $v_1, v_2, \ldots v_\ell \in \{0, 1\}$, all initially zero, and a collection of k hash functions $h_i : \{0, 1\}^* \to \{1, 2, \ldots \ell\}$, $i \in \{1, 2, \ldots k\}$. To insert a key $x \in \{0, 1\}^*$, we set $v_{h_1(x)} = v_{h_2(x)} = \cdots v_{h_k(x)} = 1$. To query for a key y, we check whether $v_{h_i(y)} = 1$ for all $i \in \{1, 2, \ldots k\}$ and return true if so. If $v_{h_i(y)} = 0$ for some $i \in \{1, 2, \ldots k\}$, we return false. Based on the number of keys one expects to store and a desired false positive rate, optimal values for ℓ and k may be selected [21].

[1] Although the ciphertexts of any probabilistic cryptosystem must be larger than the plaintexts, the message expansion can be reduced through a generalization of the Paillier cryptosystem due to Damgård and Jurik [38]. In their scheme the plaintext and ciphertext spaces are $\mathbb{Z}_{N^s}$ and $\mathbb{Z}_{N^{s+1}}^*$ for any $s \in \{1, 2, \ldots\}$. However, the constraints in this context make the original situation of $s = 1$ preferable in practice.

$\underline{\text{QUERY}}(1^\lambda, \varepsilon, S, m, n)$
Generate Paillier modulus $N = pq$
For $1 \leq i \leq |D|$:
If $w_i \in S$, $q_i \leftarrow 1$
Else $q_i \leftarrow 0$
$Q[i] \leftarrow E(q_i)$
Return $\mathsf{query} = (\varepsilon, m, n, N, Q), \mathsf{key} = (p, q)$

Figure 3.1: The QUERY algorithm.

3.2 New Constructions

We now describe the algorithms of the new private search scheme and give an analysis of their complexity and security properties. As explained in Section 1.4, our improvements to Ostrovsky-Skeith are based on allowing collisions in the main document buffer while returning additional information in one of two ways: the simple metadata construction and the Bloom filter construction. For ease of exposition, we first describe the version of the scheme using the Bloom filter construction, then give the modifications necessary to employ the simple metadata construction. Additionally, we will defer discussion of several special failure cases to the next section.

3.2.1 Query

Figure 3.1 gives the algorithm for producing the encrypted query. We assume the availability of a public dictionary of potential keywords $D = \{w_1, w_2, \ldots, w_{|D|}\}$. Constructing the encrypted query for some disjunction of keywords $S \subseteq D$ then proceeds as in the scheme of Ostrovsky and Skeith, regardless of whether the simple metadata construction or Bloom filter construction will be used. The client generates a Paillier modulus and saves its factorization as their key. For each $i \in \{1, \ldots, |D|\}$, we define $q_i = 1$ if $w_i \in S$ and $q_i = 0$ if $w_i \notin S$. The values $q_1, q_2, \ldots, q_{|D|}$ are encrypted (independently randomizing each encryption) and put in the array $Q = (E(q_1), E(q_2), \ldots, E(q_{|D|}))$. This is sent to the server along with the public key N and search parameters ε, m, and n. In Section 3.4 we give an alternative form for the encrypted queries which eliminates the public dictionary D.

3.2.2 Search (Bloom Filter Construction)

The SEARCH algorithm run by the server is shown in Figure 3.2. We begin our description of its operation by explaining the state the server must maintain.

State. In addition to the current file number i, the server must maintain three buffers as it processes the files in its stream. These buffers are hereafter referred to as the *data buffer,* the *c-buffer,* and the *matching-indices buffer* and are denoted F, C, and I respectively. Each of these is an array of elements from the ciphertext space $\mathbb{Z}_{N^2}^*$, with F and C of length ℓ_F and I of length ℓ_I. The lengths ℓ_F and ℓ_I are chosen by the server based on the parameters ε, m, and n; the considerations behind this choice are explained in Section 3.3. Each of these buffers begins with all its elements initialized to encryptions of zero, which may be computed by the server using the client's public key.[2] For simplified notation, we assume that each document is at most $\lfloor \log_2 N \rfloor$ bits and therefore fits within a single plaintext in $\mathbb{Z}_N$. For longer documents requiring s elements of $\mathbb{Z}_N$, we would let F be an $\ell_F \times s$ array and operations involving a file updating F would be performed blockwise; alternatively, the extension given in Section 3.4 could be used to allow variable-length documents.

The data buffer will store the matching files in an encrypted form which can then be used by the client to reconstruct the matching files. In particular, the data buffer will contain a system of linear equations in terms of the content of the matching files in an encrypted form. This system of equations will later be solved by the client to obtain the matching files.

The c-buffer stores in an encrypted form the number of keywords matched by each matching file. We call the number of keywords matched for a file the *c-value* of the file. The c-buffer will be used during reconstruction of the matching files from the data buffer by the client. As in the case of the data buffer, the c-buffer stores its information in the form of a system of linear equations. The client will later solve the system of linear equations to reconstruct the c-values.

The matching-indices buffer is an encrypted Bloom filter that keeps track of the indices of matching files in an encrypted form. More precisely, the matching-indices buffer will be an encrypted representation of some set of indices $\{\alpha_1, \ldots, \alpha_r\}$ where $\{\alpha_1, \ldots, \alpha_r\} \subseteq \{1, \ldots, t\}$. Here r is the number of files which end up matching the query.

Processing steps. We now detail how these buffers are updated as each file is processed. To process the ith file f, the server takes the following steps.

Step 1: Compute encrypted c-value. First, the server looks up the query array entry $Q[j]$ corresponding to each word w_j found in the file. The product of these entries is then computed. Due to the homomorphic property of the Paillier cryptosystem, this product is an encryption of the c-value of the file, i.e., the number of distinct members of S found in the file. That is,

$$\prod_{w_j \in \mathsf{words}(f)} Q[j] = E\left(\sum\nolimits_{w_j \in \mathsf{words}(f)} q_j\right) = E\left(c_i\right)$$

where $\mathsf{words}(f)$ is the set of distinct words in the ith file and c_i is defined to be $|S \cap \mathsf{words}(f)|$. Note in particular that $c_i \neq 0$ if and only if the file matches the query.

[2]Since these values need not be individually randomized, it actually suffices to initialize each to the value one, which is a valid Paillier encryption of zero under any public key.

SEARCH(query, f, buf)
If buf $\neq \perp$, Parse buf $= (i, F, C, I)$
Else initialize each element of F, C, I to $E(0)$ and let $i = 1$

/* Step 1 */
$c \leftarrow E(0)$
For $w_j \in \mathsf{words}(f)$
$c \leftarrow c \cdot Q[j] \bmod N^2$

/* Steps 2 and 3 (in parallel) */
$e \leftarrow c^f \bmod N^2$
For $1 \leq j \leq \ell_F$
If $g(i, j) = 1$
$F[j] \leftarrow F[j] \cdot e \bmod N^2$
$C[j] \leftarrow C[j] \cdot c \bmod N^2$

/* Step 4 */
For $1 \leq j \leq k$
$\ell \leftarrow h_j(i) \bmod \ell_I$
$I[\ell] \leftarrow I[\ell] \cdot c \bmod N^2$

Return buf $= (i + 1, F, C, I)$

Figure 3.2: The SEARCH algorithm.

Step 2: Update data buffer. The server computes $E(c_i)^f = E(c_i f)$ using the homomorphic property of the Paillier cryptosystem. Note that $c_i f = 0$ if f does not match the query. The server then multiplies the value $E(c_i f)$ into each location j in the data buffer where $g(i, j) = 1$ (recall that $g : \mathbb{Z} \times \mathbb{Z} \to \{0, 1\}$ is pseudo-random function chosen ahead of time as a global, public parameter). Suppose for example we are updating the third location in the data buffer with the second file, denoted f_2. Assume that the first file (f_1) was also multiplied into this location, i.e., $g(1, 3) = g(2, 3) = 1$. Each of the two files may or may not match the query. Suppose in this example that f_1 matches the query, but f_2 does not. Before processing f_2 we have $D(F[3]) = c_1 f_1 \bmod N$. After multiplying in $E(c_2 f_2)$, $D(F[3]) = c_1 f_1 + c_2 f_2 \bmod N$. But $c_2 = 0$ since f_2 does not match, so it is still the case that $D(F[3]) = c_1 f_1 \bmod N$ and the data buffer is effectively unmodified. This mechanism causes the data buffer to accumulate linear combinations of matching files while discarding all non-matching files. Note that, as shown in Figure 3.2, the server multiplies ciphertexts modulo N^2; this results in the underlying plaintexts being added modulo N. Naturally, when

several files are added modulo N, the result will "wrap around" and be mapped back into $\mathbb{Z}_N$. It is important to realize that this does not result in a loss of essential information or pose any problem to the scheme. Provided there are as many (independent) linear equations as file blocks, the value of each file block will be uniquely determined, and the client will be able to correctly recover each of the files using the EXTRACT algorithm.

Step 3: Update c-buffer. The value $E(c_i)$ is similarly multiplied into each of the locations j in the c-buffer where $E(c_i f)$ was used to update the data buffer, that is, wherever $g(i,j) = 1$.

Step 4: Update matching-indices buffer. The server then multiplies $E(c_i)$ further into a fixed number of locations in the matching-indices buffer. This is done using essentially the standard procedure for updating a Bloom filter. Specifically, we use k hash functions $h_1, \ldots, h_k$ to select the k locations where $E(c_i)$ will be applied. For optimal efficiency, the parameter k should be set to $\lfloor \frac{\ell_I \log 2}{m} \rfloor$, where m is the number of files they expect to retrieve [21]. Again, if f does not match, $c_i = 0$ so the matching-indices buffer is effectively unmodified.

3.2.3 Extract (Bloom Filter Construction)

After the server has processed some number of files t, it may return the buffers to the client, who may then obtain the results with the EXTRACT algorithm given in Figure 3.3. We now explain each stage of this algorithm.

Step 1: Decrypt buffers. The client first decrypts the values in the three buffers with key, obtaining decrypted buffers F', C', and I'.

Step 2: Reconstruct matching indices. For each $i \in \{1, 2, \ldots, t\}$, the client computes $h_1(i), h_2(i), \ldots, h_k(i)$ and checks the corresponding locations in the decrypted matching-indices buffer; if all these locations are non-zero, then i is added to the list $\alpha_1, \alpha_2, \ldots, \alpha_\beta$ of potential matching indices. Note that if $c_i \neq 0$, then i will be added to this list. However, due to the false positive feature of Bloom filters, we may obtain some additional indices. Now we may check for overflow, which occurs when the number of false positives plus the number of actual matches r exceeds ℓ_F. At this point if $\beta < \ell_F$, we add arbitrary, unique integers to the list until it is of length ℓ_F. Here the function "pick" denotes the operation of selecting an arbitrary member of a set.

Step 3: Reconstruct c-values of matching files. Given our superset of the matching indices $\{\alpha_1, \alpha_2 \ldots, \alpha_{\ell_F}\}$, the client next solves for the values of $c_{\alpha_1}, c_{\alpha_2}, \ldots, c_{\alpha_{\ell_F}}$. This is accomplished by solving the system of linear equations $A \cdot \vec{c} = C'$ for $\vec{c}$, where A is the matrix with the i,jth entry set to $g(\alpha_i, j)$, C' is the vector of values stored in the decrypted c-buffer, and $\vec{c}$ is the column vector $(c_{\alpha_i})_{i=1,\ldots,\ell_F}$. Now the exact set of matching indices $\{\alpha'_1, \alpha'_2 \ldots, \alpha'_r\}$ may be computed by checking whether $c_{\alpha_i} = 0$ for each $i \in \{1, \ldots, \ell_F\}$. Before proceeding, we replace all zeros in the vector $\vec{c}$ with ones. As an example of Step 3, suppose there are four spots in the decrypted c-buffer (i.e., $\ell_F = 4$), seven files have been processed ($t = 7$), and from Step 2 we have established the following list of potentially matching indices: $\{\alpha_1, \alpha_2, \alpha_3, \alpha_4\} = \{1, 3, 5, 7\}$. Further suppose that the matrix induced by

EXTRACT(key, buf)
/* Step 1 */
$F'[i] \leftarrow D(F[i]) \quad \forall\, 1 \le i \le \ell_F$
$C'[i] \leftarrow D(C[i]) \quad \forall\, 1 \le i \le \ell_F$
$I'[i] \leftarrow D(I[i]) \quad \forall\, 1 \le i \le \ell_I$

/* Step 2 */
$\beta \leftarrow 0$
For $1 \le i \le t$
  For $1 \le j \le k$
    $\ell \leftarrow h_j(i) \bmod \ell_I$
    If $I'[\ell] = 0$, next i
  $\alpha_\beta \leftarrow i,\ \beta \leftarrow \beta + 1$
If $\beta > \ell_F$, output "error: overflow" and exit
While $\beta < \ell_F$
  $\alpha_\beta \leftarrow \text{pick}(\mathbb{Z} \setminus \{\alpha_1, \alpha_2, \ldots, \alpha_{\beta-1}\}),\ \beta \leftarrow \beta + 1$

/* Step 3 */
$A \leftarrow \left[g(\alpha_i, j) \right]_{\substack{i \in \{1,2,\ldots,\ell_F\} \\ j \in \{1,2,\ldots,\ell_F\}}}$
If A is singular, output "error: singular matrix" and exit
$\vec{c} \leftarrow A^{-1} \cdot C'$
$\{\alpha'_1, \alpha'_2, \ldots, \alpha'_r\} = \{\alpha_1, \alpha_2, \ldots, \alpha_{\ell_F}\} \setminus \{\ \alpha_i \mid c_{\alpha_i} = 0\ \}$
$c_{\alpha_i} \leftarrow 1 \ \ \forall\, i \in \{\ \alpha_i \mid c_{\alpha_i} = 0\ \}$

/* Step 4 */
$\vec{f} \leftarrow \text{diag}(\vec{c})^{-1} \cdot A^{-1} \cdot F'$
Output $f_{\alpha'_1}, f_{\alpha'_2}, \ldots, f_{\alpha'_r}$

Figure 3.3: The EXTRACT algorithm.

the pseudo-random function g is

$$A = \left[g(\alpha_i, j) \right]_{\substack{i \in \{1,2,\ldots,\ell_F\} \\ j \in \{1,2,\ldots,\ell_F\}}} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} .$$

Then if the c-buffer decrypts to the column vector $C' = (2\ \ 3\ \ 1\ \ 3)$, we may establish the

following linear system, since $A \cdot \vec{c} = C'$.

$$\begin{aligned} c_{\alpha_1} + c_{\alpha_3} &= 2 \\ c_{\alpha_1} + c_{\alpha_2} + c_{\alpha_4} &= 3 \\ c_{\alpha_1} + c_{\alpha_4} &= 1 \\ c_{\alpha_2} + c_{\alpha_3} &= 3 \ . \end{aligned}$$

Solving, we obtain $c_{\alpha_1} = c_1 = 1$, $c_{\alpha_2} = c_3 = 2$, $c_{\alpha_3} = c_5 = 1$, and $c_{\alpha_4} = c_7 = 0$. We see now that the seventh file appeared due to a Bloom filter false positive and that there were three actual matching files ($r = 3$): f_1, f_3, and f_5.

Step 4: Reconstruct matching files. Continuing in our description of the EXTRACT algorithm with Step 4, the content of the matching files $f_{\alpha'_1}, f_{\alpha'_2}, \ldots, f_{\alpha'_r}$ may be determined by solving the linear system $A \cdot \mathrm{diag}(\vec{c}) \cdot \vec{f} = F'$, where

$$\mathrm{diag}(\vec{c}) = \begin{bmatrix} c_1 & 0 & \cdots \\ 0 & c_2 & \\ \vdots & & \ddots \end{bmatrix} \ .$$

We directly compute $\vec{f} = \mathrm{diag}(\vec{c})^{-1} \cdot A^{-1} \cdot F'$. Note that $\mathrm{diag}(\vec{c})$ is never singular because we replaced all zeros in $\vec{c}$ with ones at the end of Step 3. The content of the matching files appears as $f_{\alpha'_1}, f_{\alpha'_2}, \ldots, f_{\alpha'_r}$; the other entries in $\vec{f}$ will be zero. Continuing the example started in the description of Step 3, suppose the data buffer decrypts to $F' = (32\ 32\ 10\ 44)$. Of course, these are artificially small values; in reality they would be about 1024 bits each. Then we may solve the following system

$$\begin{aligned} f_1 + f_5 &= 32 \\ f_1 + 2f_3 + f_7 &= 32 \\ f_1 + f_7 &= 10 \\ 2f_3 + f_5 &= 44 \ , \end{aligned}$$

to determine that $f_1 = 10$, $f_3 = 11$, and $f_5 = 22$. We also find that $f_7 = 0$ as expected, since $c_7 = 0$.

Keep in mind that the linear equations for the file blocks and c-values are modulo N; that is, the values appearing the decrypted buffers F' and C' were computed modulo N as explained in the description of the SEARCH algorithm. The above example was shown using standard arithmetic for simplicity, but a system of linear equations modulo N is solved in the same way to recover the original values of each c_i and f_i.

3.2.4 The Simple Metadata Construction

Now that we have defined the version of the scheme incorporating the (more complex) Bloom filter construction, we may easily describe the differences between this version of the scheme

and the variant using the simple metadata construction. In applications where the expected number of matching documents is fixed and independent of the stream length, this latter variant is preferable since it does not require communication and storage dependent on the stream length. To produce this effect, we abandon the Bloom filter used in the matching-indices buffer and instead use the Ostrovsky-Skeith construction to store the matching indices. We briefly describe this technique below; for details (including the selection of the γ parameter) refer to [72].

Let $\ell_I = \gamma m$, where γ is selected based on the desired error bound ε. Fix a set of hash functions $h_1, h_2, \ldots, h_\gamma$. Also, let each entry in the matching-indices buffer I be a pair of ciphertexts in $\mathbb{Z}^*_{N^2}$ rather than a single ciphertext. To update I when processing the ith file in SEARCH, compute as follows.

$$\begin{aligned}&\text{For } 1 \leq j \leq \gamma:\\ &\quad \ell \leftarrow h_j(i) \bmod \ell_I\\ &\quad I[\ell][1] \leftarrow I[\ell][1] \cdot c \bmod N^2\\ &\quad I[\ell][2] \leftarrow I[\ell][2] \cdot c^i \bmod N^2\end{aligned}$$

To recover the set of matching indices in EXTRACT, the client decrypts each pair of entries in I. When a pair $I'[k][1]$ and $I'[k][2]$, $k \in \{1, \ldots \ell_I\}$ is non-zero (and not a collision), the client may recover the index of a matching file as $i = I'[k][2]/I'[k][1]$. When using this technique, the c-buffer is omitted. We may set $\ell_F = m$; otherwise, the data buffer is used as before. There are now no false positives for streams of any length.

3.3 Analysis

In this section, we consider the computation and communication complexity of both variants of our scheme and prove their security.

Computational complexity. The running time of the first client side algorithm, QUERY, is $O(|D|)$. This is exactly the same as in Ostrovsky-Skeith, in which the encrypted queries take the same form. More precisely, QUERY requires $|D|$ exponentiations and $|S| \leq |D|$ multiplications. For large dictionaries, this is a significant cost in both our scheme and Ostrovsky-Skeith; Section 3.4 presents an extension to our scheme which can greatly reduce this cost.

When using the Bloom filter construction, the SEARCH algorithm has running time $O(|\mathsf{words}(f)| + s \cdot m + \log \frac{t}{m})$ when processing a file f. Recall that $\mathsf{words}(f)$ is the set of keywords associated with that file and s is the number of plaintext blocks required to store the contents of a file. With the simple metadata construction, the SEARCH algorithm runs in time $O(|\mathsf{words}(f)| + s \cdot m + \log m)$. In either case, however, only s exponentiations are required; the rest of the computation results from multiplications.

The EXTRACT algorithm runs in time $O(s \cdot m + m^{2.376} + t \log \frac{t}{m})$ when using the Bloom filter construction or $O(s \cdot m + m^{2.376})$ with the simple metadata construction. Note that the $s \cdot m$ term is necessary to simply output the results. The $m^{2.376}$ term corresponds to solving a system of linear equations [34], and the $t \log \frac{t}{m}$ term is the time to check each possible document index against the Bloom filter. Asymptotically, these running times are neither strictly better nor strictly worse than the $O(s \cdot m \log m)$ file reconstruction time with Ostrovsky-Skeith. In practice, however, we find that file reconstruction is far faster using either of the new schemes; this is considered in detail in the next chapter.

Communication complexity (Bloom filter construction). We now consider communication complexity, beginning with the scheme employing the Bloom filter construction. In particular, we will show that given a desired success probability bound $1-\varepsilon$, if the number of matching documents is at most m and each is of length s, then by using communication and storage overhead $O(s \cdot m + m \log \frac{t}{m})$, our scheme will enable the user to correctly reconstruct all the matching documents from a stream of t documents with probability at least $1 - \varepsilon$.

In order to perform the analysis to demonstrate the above point, we first consider the failure cases where the user will be unable to reconstruct the matching documents. From the reconstruction procedure, we can see that the client fails to reconstruct the matching files when the two systems of linear equations $A \cdot \vec{c} = C'$ and $A \cdot \mathrm{diag}(\vec{c}) \cdot \vec{f} = F'$ cannot be correctly solved. This failure only happens in two cases:

1. The matrix A is singular. In this case, we will not be able to compute A^{-1} and solve the system of linear equations.
2. There are more than $\ell_F - m$ false positives when the set of matching indices is computed using the Bloom filter. Specifically, if in Step 2 in the EXTRACT procedure, the number of matching indices β reconstructed from the Bloom filter I' is greater than ℓ_F, then we have more variables than the number of linear equations and thus we will not be able to solve the system of linear equations $A \cdot \vec{c} = C'$.

We show below that by picking the parameters ℓ_F and ℓ_I correctly, we can guarantee that the probability of the above two failure cases can be bounded to be below ε. We demonstrate this by proving the following three lemmas.

Lemma 3.3.1. *For a given $0 < \varepsilon < 1$, there exists $n = o(\log \frac{1}{\varepsilon})$, such that for any $n' > n$, an $n' \times n'$ random $(0,1)$-matrix is singular with probability at most ε.*

Proof. Note that an $n \times n$, random (0,1)-matrix is singular with negligible probability in n. This was first conjectured by Erdös and was proven in the 1960's [57]. The specific bound has since been improved several times, recently reaching $O\left(\left(\frac{3}{4} + o(1)\right)^n\right)$ [54, 83, 84]. Thus, it is easy to see that the above lemma holds. □

Lemma 3.3.2. *Let $G : \mathcal{K}_G \times \mathbb{Z} \times \mathbb{Z} \to \{0,1\}$ be a $(\omega_t, \omega_q, \frac{\varepsilon}{8})$-secure pseudo-random function family. Let $g = G_k$, where $k \xleftarrow{R} \mathcal{K}_G$. Let $\ell_F = o(\log \frac{1}{\varepsilon})$ such that an $\ell_F \times \ell_F$ random $(0,1)$-matrix is singular with probability at most $\frac{\varepsilon}{4}$. Then the matrix*

$$A = \Big[\, g(i,j) \,\Big]_{\substack{i=1,\ldots,\ell_F \\ j=1,\ldots,\ell_F}}$$

is singular with probability at most $\frac{\varepsilon}{2}$.

Intuitively, this lemma bounds the failure probability that the matrix A is singular. We provide the proof in Appendix C. Additionally, we note that for a given constant ε the size of the ℓ_F will be linear in m.

Lemma 3.3.3. *Given $\ell_F > m + 8\ln(\frac{2}{\varepsilon})$, let $\ell_I = O(m \log \frac{t}{m})$ and assume the number of matching files is at most m out of a stream of t. Then the probability that the number of reconstructed matching indices β is greater than ℓ_F is at most $\frac{\varepsilon}{2}$.*

Given the false positive rate of a Bloom filter, the proof is straightforward and also available in Appendix C. Together, Lemma 3.3.2 and Lemma 3.3.3 provide the primary result:

Theorem 3.3.4. *If $\ell_F = o(\log \frac{1}{\varepsilon}) + O(m)$, $\ell_F > m + 8\ln(\frac{2}{\varepsilon})$, $\ell_I = O(m \log \frac{t}{m})$, and $G : \mathcal{K}_G \times \mathbb{Z} \times \mathbb{Z} \to \{0,1\}$ is a $(\omega_t, \omega_q, \frac{\varepsilon}{8})$-secure pseudo-random function family, then when the number of matching files is at most m in a stream of t, the new scheme using the Bloom filter construction guarantees that the client can correctly reconstruct all matching files with probability at least $1 - \varepsilon$.*

Proof. By Lemma 3.3.2, the probability that the matrix A is singular is at most $\frac{\varepsilon}{2}$. By Lemma 3.3.3, the probability that the reconstruction of the matching indices will yield more than ℓ_F matching indices is at most $\frac{\varepsilon}{2}$. Since these are the only two failure cases as explained earlier, the total failure probability, the probability that the client would fail to reconstruct the matching files, is at most ε. □

Communication complexity (simple metadata construction). We now consider the complexity in the case of using the simple metadata construction.

Theorem 3.3.5. *If $\ell_F = o(\log \frac{1}{\varepsilon}) + O(m)$, $\ell_F > m + 8\ln(\frac{2}{\varepsilon})$, $\ell_I = O(m(\log m + \log \frac{1}{\varepsilon}))$, and $G : \mathcal{K}_G \times \mathbb{Z} \times \mathbb{Z} \to \{0,1\}$ is a $(\omega_t, \omega_q, \frac{\varepsilon}{8})$-secure pseudo-random function family, then when the number of matching files is at most m, the new scheme using the simple metadata construction guarantees that the client can correctly reconstruct all matching files with probability at least $1 - \varepsilon$.*

Proof. Briefly, the argument for Theorem 3.3.4 may be applied again, except that we no longer need Lemma 3.3.3. Instead, we refer to the analysis in [72] that demonstrates that the probability of an overflow in the alternative matching-indices buffer may be bounded below ε with $\ell_I = \gamma m$ where $\gamma = O(\log m + \log \frac{1}{\varepsilon})$, producing an overall communication and storage complexity of $O(m \log m)$. □

Note that our scheme still produces a constant factor improvement over Ostrovsky-Skeith in this case. If each file requires s plaintext blocks (i.e., is of length at most $s \cdot \lfloor \log_2 n \rfloor$ bits), then we reduce communication and storage by a factor of approximately $\log(sm)$ for large files. This is accomplished by retrieving the bulk of the content through the efficient data buffer and only retrieving document indices through the less efficient matching-indices buffer.

Security. The security of the proposed scheme (in both variants) is straightforward. Intuitively, since the server is only provided with an array of encryptions of ones and zeros, the scheme should be as secure as the underlying cryptosystem.

Theorem 3.3.6. *If the Paillier cryptosystem is semantically secure, then the proposed private searching scheme is semantically secure according to the definition in Section 1.4.*

This proof, which is identical to the one for the Ostrovsky-Skeith scheme, is also provided in Appendix C. Since the proof depends only on the form of the encrypted query, the variant of the scheme which will be used is irrelevant. Note that this theorem establishes security based on the decisional composite residuosity assumption (DCRA), since the Paillier cryptosystem has been proven semantically secure based on that assumption [74].

3.4 Extensions

Here we describe a number of extensions to the proposed system which provide additional features.

Compressing the Bloom filter. For security it will generally be necessary to use a modulus N of at least 1024 bits (e.g., as required by the standards ANSI X9.30, X9.31, X9.42, and X9.44 and FIPS 186-2) [79]. If the Bloom filter construction is used, the fact that the c-values will never approach 2^{1024} reveals that most of its space is wasted. A simple technique allows improved usage of this space. If we assume that the sums of c-values appearing in each location in I will be less than 2^{16}, for example, we may use each group element to represent $\frac{n}{16}$ array entries. In the case of $n = 1024$, this reduces the size of I by a factor of 64. When we need to multiply a value $E(c)$ into the Bloom filter in the SEARCH algorithm, we use the following technique. To multiply it into the ith location in I, we let $i_1 = \lfloor \frac{i}{64} \rfloor$ and $i_2 = i \bmod 64$. Then we compute

$$I[i_1] \leftarrow I[i_1] \cdot E(c)^{2^{16 i_2}}$$

which has the result of shifting c into the i_2th 16-bit block within the group element in $I[i_1]$. After the client decrypts I, it may simply break up each element into 64 regions of 16 bits. However, this space savings comes at an additional computation cost. The server will need to perform k additional modular exponentiations for each file it processes.

Hashing keywords. In some applications, a predetermined set of possible keywords D may be unacceptable. Many of the strings a user may want to search for are obscure (e.g., names of particular people or other proper nouns) and including them in D would already reveal too much information. Since the size of encrypted queries is proportional to $|D|$, it may not be feasible to fill D with, say, every person's name, much less all proper nouns.

In such applications an alternative form of encrypted query may be used. Eliminating D, we allow S to be any finite subset of Σ^*, where Σ is some alphabet. Now in QUERY, we pick a length ℓ_Q for the array Q and initialize each element to $E(0)$. Then for each $w \in S$, we use a hash function $h : \Sigma^* \to \{1, \ldots, \ell_Q\}$ to select a location $h(w)$ in Q and set $Q[h(w)] \leftarrow E(1)$. As before we rerandomize each encryption of zero and one. To process the ith file f_i in SEARCH, the server may now compute $E(c_i) = \prod_{w \in \mathsf{words}(f_i)} Q[h(w)]$. The rest of the scheme is unmodified. Using this extension, it is possible for a file f_i to spuriously match the query if there is some word $w' \in \mathsf{words}(f_i)$ such that $h(w') = h(w)$ for some $w \neq w'$ in S. The possibility of such false positives is the key disadvantage of this approach.

An advantage of this alternative approach, however, is that it is possible to extend the types of possible queries. Previously only disjunctions of keywords in D were allowed, but in this case a limited sort of conjunction of strings may be achieved. To support queries of the form "$w_1\ w_2$" where $w_1, w_2 \in \Sigma^*$, we change the way each $\mathsf{words}(f_i)$ is derived from the corresponding file f_i. In addition to including each word found in the file f_i, we include all adjacent pairs of words in $\mathsf{words}(f_i)$ (note that this approximately doubles the size of $\mathsf{words}(f_i)$). It is easy to imagine further extensions along these lines. In particular, it is possible to match against binary data by simply including blocks of the contents of f_i in $\mathsf{words}(f_i)$. See Section 4.2 for a discussion of the size and computation time corresponding to various query sizes in practical settings.

Arbitrary length files. In applications where the files are expected to vary significantly in length, an unacceptable amount of space may be wasted by setting an upper bound on the length of the files and padding smaller files to that length. Here we describe a modification to the scheme which eliminates this source of inefficiency by storing each block of a file separately. For convenience, we describe it in terms of the version of the scheme employing the Bloom filter; applying this technique to the other variant is straightforward.

In this extension QUERY takes two upper bounds on the matching content. We let m be an upper bound on the number of matching files and n be an upper bound on the total length of the matching files, expressed in units of Paillier plaintext blocks. As before, the c-buffer is of length $O(m)$ and the matching-indices buffer is of length $O(m \log \frac{t}{m})$ (or, using the alternative construction given in Section 3.2.4, $O(m \log m)$). The data buffer is now set to length $O(n)$, and each entry in the data buffer is now a single ciphertext rather than an array fixed to an upper bound on the length of each file. We introduce a new buffer on the server called the *length buffer,* which is an array L set to length $O(m)$. Intuitively, the length buffer will be used to store the length of each matching file, and the data buffer will now be

used to store linear combinations of individual blocks from each file rather than entire files.

We briefly describe how this is accomplished in more concrete terms. Replace the corresponding portion of SEARCH with the following, where $\ell_C = O(m)$ is the length of the c-buffer and length buffer, $\ell_F = O(n)$ is the length of the data buffer, $\hat{g} : \mathbb{Z}^3 \to \{0, 1\}$ is an additional pseudo-random function, d_i is the length of the ith file in the stream, and the d_i blocks of the file are denoted $f_{i,1}, f_{i,2}, \ldots, f_{i,d_i}$.

$e \leftarrow c^{d_i} \bmod N^2$
For $1 \leq j \leq \ell_C$:
  If $g(i, j) = 1$
    $C[j] \leftarrow C[j] \cdot c \bmod N^2$
    $L[j] \leftarrow L[j] \cdot e \bmod N^2$

For $1 \leq j_1 \leq d_i$:
  $e \leftarrow c^{f_{i,j_1}} \bmod N^2$
  For $1 \leq j_2 \leq \ell_F$:
    If $\hat{g}(i, j_1, j_2) = 1$
      $F[j_2] \leftarrow F[j_2] \cdot e \bmod N^2$

The client may use a modified version of EXTRACT to recover the matching files. As before, the matching-indices buffer I is used to determine a superset of the indices of matching files, and a matrix A of length ℓ_C is constructed based on these indices using g. The vector $\vec{c}$ is again computed as $\vec{c} \leftarrow A^{-1} \cdot C'$. The client next computes the lengths of the matching files as $\vec{d} \leftarrow \mathrm{diag}(\vec{c})^{-1} \cdot A^{-1} \cdot L'$. If $\sum_i d_i > \ell_F$, the combined length of the files is greater than the prescribed upper bound and the client aborts. Otherwise, the data buffer now stores a system of $\ell_F \geq n$ linear equations in terms of the individual blocks of the matching files. Briefly, the blocks may be recovered by constructing a new matrix $\hat{A}$, filling its entries by evaluating $\hat{g}$ over the indices of the blocks of the matching files. The blocks of the matching files are then computed as $\vec{f} \leftarrow \mathrm{diag}(\vec{c}')^{-1} \cdot \hat{A}^{-1} \cdot F'$, where $\vec{c}'$ is as $\vec{c}$ but with the ith entry repeated d_i times.

Using this extension, space may be saved if the matching files are expected to vary in size. Some information about the number expected to match and their total size is still needed to set up the query, but the available space may now be distributed arbitrarily amongst the files.

Merging parallel searches. Another extension makes multiple server, distributed searches possible. Suppose a collection of servers each have their own stream of files. The client wishes to run a private search on each of them, but does not wish to separately download and decipher a full size buffer of results from each. Instead, the client wants the servers to remotely merge their results before returning them.

This can be accomplished by simply having each server separately run the search algorithm, then multiplying together (element by element) each of the arrays of resulting

ciphertexts. This merging step can take place on a single collecting server, or in a hierarchical fashion. A careful investigation of the algorithms reveals that the homomorphism ensures the result is the same as it would be if a single server had searched the documents in all the streams. Care must be taken, however, to ensure the uniqueness of the document indices across multiple servers. This can be accomplished by, for example, having servers prepend their IP address to each document index. Also, it is of course necessary for the buffers on each server to be of the same length.

Note that if the client splits its query and sends it to each of the remote servers, a different set of keywords may be used for each stream. Alternatively, a server may split a query to be processed in parallel for efficiency without the knowledge or participation of the client.

Chapter 4

Practical Feasibility

In the preceding two chapters, we introduced two new cryptographic schemes and showed that their asymptotic costs are acceptable. In this chapter, we investigate these costs in more detail to determine whether our proposals are feasible in practical scenarios. We begin by giving further consideration to our signatures of reputation.

4.1 Space Requirements for Signatures of Reputation

Because our scheme is the first proposed for signatures of reputation, there is no previous work to which we can compare its efficiency. Instead, we ask whether its use is feasible at all using present technology. In particular, to our knowledge, no one has yet attempted to implement the Groth-Sahai non-interactive proof system [49], so its costs deserve further attention. Note that Groth and Sahai present three variations of their scheme. Because our scheme operates in prime order groups and already requires the decisional linear assumption, we use the the version of the Groth-Sahai scheme based on that assumption.

To make our analysis as concrete as possible, we assume an implementation of the scheme will employ the Pairing Based Cryptography (PBC) library [63], a widely used library for high performance computations in elliptic curve groups with associated pairings. Recall that our scheme requires a bilinear map $e : \mathbb{G} \times \widehat{\mathbb{G}} \to \mathbb{G}_T$ between groups of prime order p. The first four rows of Table 4.1 give the size of elements in each of these groups (and the group of exponents, $\mathbb{Z}_p$) as they are stored by PBC. We assume the use of the curve D159 provided with the library, an MNT curve of embedding degree six [68]. Based on these sizes, a careful account of the group elements returned by each algorithm of our scheme reveals the sizes given in the remaining rows of the table.

As shown by the table, the space usage of one-time pseudonyms and votes is low. Signatures of reputation, however, are quite costly, requiring nearly 0.5 MB to demonstrate possession of each vote. While this is well within the capabilities of modern computing and communication infrastructure, it is costly enough that we would not expect the scheme to be

Object	Size
Element of $\mathbb{G}$	21 bytes
Element of $\widehat{\mathbb{G}}$	61 bytes
Element of $\mathbb{G}_T$	120 bytes
Element of $\mathbb{Z}_p$	20 bytes
One-time pseudonym	25.5 KB
Vote	49.6 KB
Signature of reputation k	21.3 KB + $k \cdot 456.4$ KB

Table 4.1: Sizes of various objects within the scheme for signatures of reputation.

used in casual situations where it provides only a minor benefit. Developing a more efficient and thus more broadly applicable scheme would be useful future work.

While signatures of reputation are large by today's standards, even for small numbers of votes, the size of ε-sound signatures does scale well asymptotically. Figure 4.1 compares signature size for several values of ε as the number of votes grows very large. After around five thousand votes, the size of the signatures becomes essentially constant. In our calculations we assumed the use of SHA-1 to construct the Merkle hash tree and picked challenge set sizes which correspond to the level of security provided by the overall scheme.

4.2 Private Stream Searching Performance

We now turn to an analysis of our other primary contribution, the private stream searching scheme. To better assess its applicability in practical scenarios, we detail its costs in a realistic application. Specifically, we consider the case of making a private version of Google's existing News Alerts service[1] using the new scheme. Use of our construction to retrieve the votes of the scheme for signatures of reputation would be similar but less demanding.

According to the Google News website, their web crawlers continuously monitor approximately 4,500 news websites. These include major news portals such as CNN along with many websites of newspapers, local television stations, and magazines. The alerts service allows a user to register keywords of interest with Google's servers. News articles are checked for the search terms as they arise, and the user receives periodic emails listing matches. In this setting, we analyze four aspects of the resources necessary to conduct the searches without revealing the search terms to Google: the size of the query sent to the server (s_q), the size of the storage buffers kept by the server while running the search and eventually transmitted back to the client (s_b), the time for the server to process a single file in its stream (t_p), and the time for the client to decrypt and recover the original matching files from the information it receives from the server (t_r). Due to the potential sensitivity of search keywords, we will

[1]See `http://www.google.com/alerts/`.

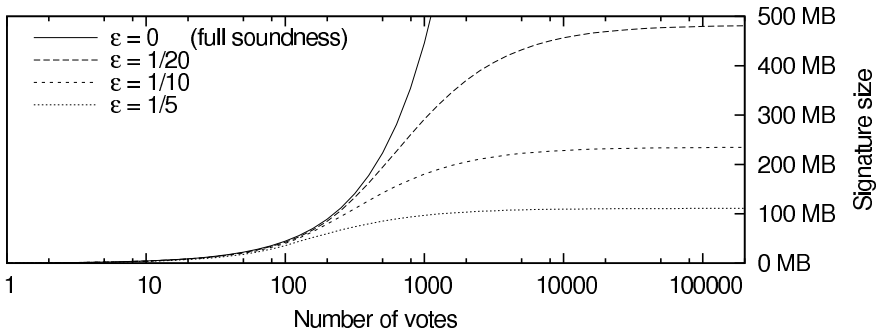


Figure 4.1: Signatures of reputation with large numbers of votes.

not use a public dictionary and we instead assume the use of the hashing extension described in Section 3.4.

We assume that the client is able to estimate an upper bound m on the number of files in the stream of t that will match the query. In situations where m may be more difficult to estimate or bound, an alternative method for selecting it may be used, at the expense of a small loss in privacy. Specifically, the server may assist the client in selecting m by computing and returning encrypted c-values for a series of files during some initial monitoring period. After decrypting the c-values, the client will know exactly how many files matched their query during the monitoring period and use this information to select m before beginning the normal stream search. While one iteration of this process may provide the server with some information about possible keywords in the query, a full dictionary attack will not be possible due to the required client participation in decrypting any c-values.

Query space. If we assume a 1024-bit Paillier key, then the encrypted query Q is $256\ell_Q$ bytes, since each element from the set of ciphertexts $\mathbb{Z}^*_{N^2}$ is $\frac{\lceil \log_2 n \rceil}{4}$ bytes, where n is the public modulus. The smaller ℓ_Q is, the more files will spuriously match the query. Specifically, treating the hash function used in constructing the query as a random oracle, we obtain the following formula for the probability θ that a non-matching file f_i will nevertheless result in a non-zero corresponding $E(c)$ (rearranged on the right to solve for ℓ_Q).

$$\theta = 1 - \left(1 - \frac{|S|}{\ell_Q}\right)^{|\mathsf{words}(f_i)|} \qquad \ell_Q = \frac{|S|}{1 - (1-\theta)^{\frac{1}{|\mathsf{words}(f_i)|}}}$$

We performed a sampling of the news articles linked by Google News and found that the average distinct word count is about 540 per article. This produces the false positive rates for several query sizes listed in Table 4.2. The first column specifies a rate of spurious matches θ and the second column gives the size s_q of the minimal Q necessary to achieve that rate for a single keyword search. Additional keywords increase s_q proportionally (e.g., $|S| = 2$ would double the value of s_q). It should be apparent that this is a significant cost;

False positive rate	Basic s_q	Reduced s_q
0.1	1.3 MB	0.3 MB
0.01	13.1 MB	3.6 MB
0.001	132.8 MB	36.6 MB

Table 4.2: Encrypted query sizes necessary to ensure various false positive rates.

in fact, it turns out that s_q is the most significant component in the total resource usage of the system under typical circumstances.

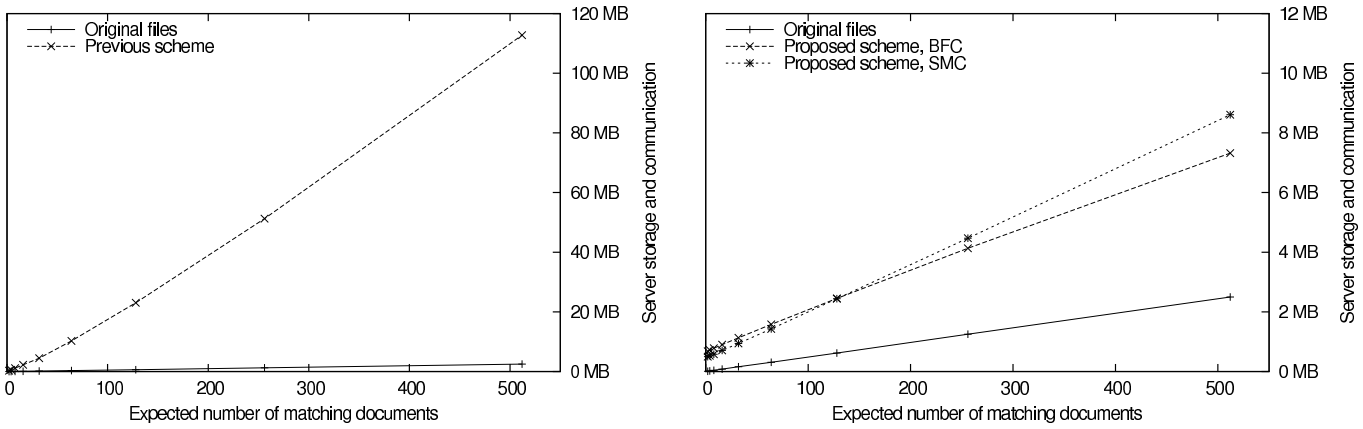
Two measures may be taken to reduce this cost. First, a majority of distinct words occurring in the text of a news article are common English words that are not likely to be useful search terms. Given this observation, the client may specify that the server should ignore the most commonly occurring words when processing each file. A review of the 3000 most common English words[2] confirms that none are likely to be useful search terms. Ignoring those words reduces the average distinct word count in a news article to about 200.

The second consideration in reducing s_q is that a smaller Paillier key may be acceptable. While 1024 bits is generally accepted to be the minimum public modulus secure for a moderate time frame (e.g., as required by the standards ANSI X9.30, X9.31, X9.42, and X9.44 and FIPS 186-2) [79], in some applications only short term guarantees of secrecy may be required. Also, a compromise of the Paillier key would not immediately result in the revelation of S. Instead, it would allow the adversary to mount a dictionary attack, checking potential members of S against Q (which will also yield many false positives). Given this consideration, if the client decides a smaller key length is acceptable, s_q will be reduced. The third column in Table 4.2 gives the size of the encrypted query using a 768-bit key and pruning out the 3000 most common English words from those searched.

Despite the significant cost of s_q in our system, the cost to obtain a comparable level of security is likely to be much greater in the system of Ostrovsky and Skeith. In that case $s_q = 256|D|$, where $|D|$ is the set of all possible keywords that could be searched. In order to reasonably hide $S \subseteq D$, $|D|$ may have to be quite large. For example, if we wish to include names of persons in S, in order to keep them sufficiently hidden we must include many names with them in D. If D consists of names from the U.S. population, s_q will be over 70 GB. It is important to emphasize, however, that the system is not truly stream length independent when using the keyword hashing technique. Checking longer streams may result in more false positives, but when using a public dictionary as in Ostrovsky and Skeith, no false positives are possible.

Server storage. We now turn to the size of the buffers maintained by the server during the search and then sent back to the client. This cost, s_b, is both a storage requirement of

[2]Based on the British National Corpus: `http://www.natcorp.ox.ac.uk/`.



(a) Ostrovsky-Skeith.

(b) New scheme.

Figure 4.2: Server storage and server to client communication in the proposed and previous schemes. Note difference in scale.

the server conducting the search and a communication requirement at the end of the search. We assume fixed length files for this application rather than employing the extension for arbitrary length files. In Bloom filter variant of the new scheme, to store the data buffer F, the c-buffer C, and the matching-indices buffer I, the server then uses

$$s_b = 256(s\ell_F + \ell_F + \ell_I).$$

bytes, where s is the number of number of plaintexts from $\mathbb{Z}_N$ required to represent an article and we assume the use of a 1024-bit key.

The client will specify ℓ_F and ℓ_I based on the number of documents they expect their search to match in one period and the desired correctness guarantees. In the case of Google news, we may estimate that each of the 4,500 crawled news sources produces an average of 30 articles per day. If the client retrieves the current search results four times per day, then the number of files processed in each period is $t = 33,750$. Now the client cannot know ahead of time how many articles will match their query, so they instead estimate an upper bound m. Based on this estimate, the analysis in Section 3.3 may be used to select values for ℓ_F and ℓ_I that ensure the probability of an overflow is acceptably small. In these experiments, we determined the minimum values for ℓ_F and ℓ_I empirically.

A range of desired values of m were considered and the results are displayed in Figure 4.2 (b). In each case, ℓ_F and ℓ_I were selected so that the probability of an overflow was less than 0.01. In computing this probability, we treated the number of documents which actually match the query as a binomial random variable with t trials and rate parameter $\frac{m}{t}$, as would be the case if each matches with some probability independent of the others. Also,

Files to retrieve m	t_p with 768-bit key	t_p with 1024-bit key
2	359 ms	600 ms
8	362 ms	600 ms
32	373 ms	603 ms
128	420 ms	617 ms
512	593 ms	669 ms

Table 4.3: Server processing time.

the spurious match rate θ was taken to be 0.001, and the news articles were considered to be 5 KB in size (text only, compressed). Note that s_b is linear with respect to the size of the matching files. More specifically, Figure 4.2 (b) reveals that s_b is about 2.5 times the size of the matching files. We also show the result of using the simple metadata construction with the new scheme, which performs about as well as the Bloom filter construction for small searches but becomes less efficient with larger numbers of documents. For comparison, the data stored by the server and returned to the client using the Ostrovsky-Skeith scheme for private searching in this scenario is shown in Figure 4.2 (a).[3] Note that this graph differs in scale from Figure 4.2 (b) by a factor of ten.

To summarize, in the proposed system s_b ranges from about 680 KB to about 7.3 MB when the expected number of matching files ranges from 2 to 512 and the overflow rate is held below 0.01. In the Ostrovsky-Skeith scheme, s_b would range from about 280 KB to 110 MB.

File stream processing time. Next we consider the time t_p necessary for the server to process each file in its stream. This is essentially determined by the time necessary for modular multiplications in $\mathbb{Z}_{N^2}^*$ and modular exponentiations in $\mathbb{Z}_{N^2}^*$ with exponents in $\mathbb{Z}_N$. To roughly estimate these times, benchmarks were run on a modern workstation. The processor was a 64-bit, 3.2 GHz Pentium 4. We used the GNU Multiple Precision Arithmetic Library (GMP), a library for arbitrary precision arithmetic that is suitable for cryptographic applications. With 768-bit keys, multiplications and exponentiations took 3.9μs and 6.2 ms respectively. With 1024-bit keys, the times increased to 6.3μs and 14.7 ms.

The first step in processing the ith file in the SEARCH procedure is computing $E(c)$; this takes $|\mathsf{words}(f_i)| - 1$ multiplications. We again assume $|\mathsf{words}(f_i)| = 540$ as discussed previously. Computing $E(cf_i)$ requires s modular exponentiations. Updating the data buffer requires an average of $s \cdot \frac{\ell_F}{2}$ modular multiplications, updating the c-buffer requires another

[3]The paper describing this system did not explicitly state a minimum buffer length for a given number of files expected to be retrieved and a desired probability of success, but instead gave a loose upper bound on the length. Rather than using the bound, we ran a series of simulations to determine exactly how small the buffer could be made while maintaining an overflow rate below 0.05.

Key length	m	Decryption time	Inversion time	Total time
768	2	14 s	<0.1 s	14 s
768	8	15 s	<0.1 s	15 s
768	32	23 s	<0.1 s	23 s
768	128	54 s	1.4 s	55 s
768	512	2.7 m	1.8 m	4.5 m
1024	2	23 s	<0.1 s	23 s
1024	8	26 s	<0.1 s	26 s
1024	32	38 s	<0.1 s	38 s
1024	128	1.4 m	21 s	1.8 m
1024	512	4.4 m	2.9 m	7.3 m

Table 4.4: Time (in seconds and minutes) necessary to recover the original documents from the returned results.

$\frac{\ell_F}{2}$ multiplications, and updating the matching-indices buffer requires $k = \lfloor \frac{\ell_I \log 2}{m} \rfloor$ multiplications. The time necessary for these steps is given for several values of m in Table 4.3. The majority of t_p is due to the s modular exponentiations. Since the Ostrovsky-Skeith scheme requires the same number of modular exponentiations, the processing time for each file would be similar.

File recovery time. Finally, we consider the time necessary for the client to recover the original matching files after a period of searching, t_r. This time is composed of the time to decrypt the returned buffers and the time to setup and solve a system of linear equations, producing the matching documents. A decryption requires 1536 modular multiplications with a 1024-bit key and 1152 with a 768-bit key [38]. The times necessary to decrypt the buffers are given in the third column of Table 4.4. This time is typically less than a minute, but can take almost five minutes with many files.

The most straightforward way to solve the system of linear equations is by performing LUP decomposition over $\mathbb{Z}_N$. Although LUP decomposition of an $n \times n$ matrix is $\Theta(n^3)$, practical cases are quite feasible. A naive implementation resulted in the benchmarks shown in the fourth column of Table 4.4. The total time to recover the matching files is given in the final column of Table 4.4.

Although the time spent in matrix inversions is a significant additional cost of the new scheme over Ostrovsky-Skeith, it is more than offset by the reduced buffer size and resulting reduction in decryption time. In Ostrovsky-Skeith, the times to decrypt the buffer returned to the client in this scenario range from 6.79 seconds for $m = 2$ to 45.5 minutes for $m = 512$, using a 768-bit key. With a 1024-bit key, the buffer decryption times range from 10.8 seconds to 1.21 hours.

4.3 Summary

Through a series of detailed calculations, we have determined that the cryptographic schemes of the previous two chapters are feasible with current technology, although in some respects they are costly. In the case of our scheme for signatures of reputation, all requirements are low except the space required by the signatures. This cost is significant: about 0.5 MB per vote. Nevertheless, we believe our proposal is valuable as the first scheme solving a problem of this type and leave the development of a more efficient system to future work.

The analysis of our scheme for private stream searching has shown that it could be applied in scenarios not previously practical. In particular, we have considered the case of conducting a private search on essentially all news articles on the web as they are generated, estimating this number to be 135,000 articles per day. In order to establish the private search, the client has a one time cost of approximately 10 MB to 100 MB in upload bandwidth. Several times per day, they download about 500 KB to 7 MB of new search results, allowing up to about 500 articles per time interval. After receiving the encrypted results, the client's PC spends under a minute recovering the original files, or up to about 7 minutes if many files are retrieved. To provide the searching service, the server keeps about 500 KB to 7 MB of storage for the client and spends roughly 500 ms processing each new article it encounters. In this scenario, the previous scheme would require up to twelve times the communication and take up to four times as long for the client to recover the results.

Chapter 5

Internet-Scale Author Identification

Previous research has resulted in methods for decoupling identifying information such as IP addresses from a user's communications at the network level [40], and the cryptographic schemes proposed in the preceding chapters are similarly designed to remove the need for conventional identity at the application level. If all of these techniques were successfully employed together, would any method of collecting a user's actions into an identifying history remain? One possibility comes to mind: guessing the source of information posted online based on nothing but its human-readable content. After all, any manually generated material will inevitably reflect some characteristics of the person who authored it, and in some scenarios these characteristics may be enough to determine whether two pieces of content were produced by the same person. For example, perhaps some individual is prone to several specific spelling errors or has other recognizable idiosyncrasies. If enough material is linked together through such characteristics, it may become personally identifiable, just as if it had all been published under a single pseudonym. In this chapter, we present the first investigation into the possibility of this process taking place on a sufficiently large-scale to constitute a widespread threat to anonymity.

We focus our investigation on textual content, which is by far the most common type of human-generated material and is well-suited to automated analysis. The problem of linking texts by author presents itself in two forms: clustering and classification. In the former case, one begins with a set of individual texts and attempts to group them by author. In the latter, a set of example texts are already grouped by author, and the aim is to match one or more new, unlabeled texts with one of the existing groups. Both tasks are similar at a technical level, and many approaches to one can be applied to the other. At our disposal are the techniques of machine learning and stylometry, the study of author-correlated features of prose for the purpose of authorship attribution.

Our approach in gauging the risk of widespread linking of textual content is, essentially, to try it ourselves and see whether we succeed. Of course, this approach can only confirm the possibility of such an attack—it cannot demonstrate its impossibility. Thus, our goal is to establish the first lower bound on the severity of this type of risk. To this end, we

have assembled large dataset of written content posted on the Internet by 100,000 users. After extracting appropriate features from the sample texts, we apply conventional machine learning algorithms to the task of matching the texts by author, simulating an attempt to determine the author of some anonymously published content of interest.

Written content is published online in many forms, including message boards, wikis, and various forms of live chat, such as IRC. In conducting our experiments, we chose blogs as the source of textual content for several reasons. First, at the present time, they constitute one of the most commonly used media of expression in written form, and large numbers of blogs are readily available for assembly into an experimental dataset. Writing in blog format generally includes longer passages of prose than message boards, and blogs are a common choice for political expression, which is especially sensitive to issues of personal identifiability.

Numerous examples exist of attempts to determine the author of an anonymously published blog. In 2009, fashion model Liskula Cohen filed a $3 million defamation lawsuit against the anonymous author of a blog for calling her an insulting name. She eventually succeeded in obtaining a court order demanding that Google, which owns the Blogger website hosting the blog, reveal the identity of the blog's owner. Google complied, revealing the author's name as Rosemary Port [14]. In the same year, an anonymous blogger criticized the $300,000 salary and other perks received by Mac Brunson, pastor of a large church in Jacksonville, Florida. Brunson asked detective Robert Hinson of the local sheriff's department (and member of Brunson's church) to investigate the blog and its author. Although the criticisms published by the blog were not in any way threatening, Hinson managed to obtain a subpoena requiring Google to reveal the author's identity. Again, Google complied, revealing the blogger as Thomas A. Rich. After forwarding his name to Brunson, the sheriff's department ceased their investigation without ever charging the author with a crime or explaining the motivation for their investigation [22].

Both of these efforts relied on the author revealing their name or IP address to a service provider, who in turn passed on that information. A careful author need not register for a service with their real name, and tools such as Tor can be used to hide their identity at the network level [40]. But if authors can be identified based on nothing but a passive comparison of the content they publish to other content found on the web, no networking tools can possibly protect them—this is the threat we aim to evaluate.

5.1 Related Work

Attempts to identify the author of a text based on the style of writing long predate computers. Originally, stylometry arose in the context of literary criticism and consisted of manual analysis by experts. The statistical approach to stylometry in the computer era was pioneered in 1964 by Mosteller and Wallace, who identified the authors of the disputed Federalist Papers [69]. While their results merely confirmed what a growing consensus of historians had already concluded, the automated nature of their methods illustrated their power relative

to previous approaches, which relied on specialized knowledge of the subject matter and potential authors of a text. Modern techniques for author identification combine stylometric features with standard machine learning algorithms and have attempted to classify texts with up to 300 candidate authors [1, 65, 39].

Stylometry has been used to attribute authorship in domains other than text, such as music [6] and code [77], which also have grammars and other linguistic features shared with natural language. Other forensic tasks such as identifying the file type of a binary blob [61] use similar techniques, although the models are simpler than linguistic stylometry. Plagiarism detection is another application of authorship attribution; however, it emphasizes content over style [66]. Juola has a survey of authorship attribution, not limited to stylometry [52].

Little work has been done to investigate the privacy implications of stylometry, however. Several researchers have considered whether the author of an academic paper under blind review might be identified solely from the citations [50, 19]. However, to our knowledge, no prior work has attempted to perform author identification at anything approaching "Internet scale" in terms of the number of authors. Our work aims to determine whether such a widespread threat exists rather than an attack targeted at some small group of individuals. Other research in this area has investigated manual and automated techniques authors may employ to make their writing more resistant to identification [20, 53].

5.2 Experimental Approach

In this section, we discuss how a real attack would work, the motivation behind our experimental design, and what we hope to learn from the experiments.

Primarily, we wish to simulate an attempt to identify the author of an anonymously published blog. If the author is careful to avoid revealing their IP address or any other explicit identifier, their adversary may turn to an analysis of writing style. By comparing the posts of the anonymous blog with a corpus of samples taken from many other blogs throughout the Internet, the adversary may hope to find a second, more easily identified blog by the same author. To have any hope of success, the adversary will need to compare the anonymous text to far more samples than could be done manually, so they instead extract numerical features and conduct an automated search for statistically similar samples.

Of course, this approach is unlikely to produce any conclusive proof of a match. Instead, we imagine the adversary's tools returning a list of the most similar possibilities for manual followup. A manual examination may incorporate several characteristics that the automated analysis does not, such as the location of the author.[1] Alternatively, a powerful adversary such as a government censor may require Google or another popular blog host to reveal the login times of the top suspects, which could be correlated with the timing of posts on

[1]For example, if we were trying to identify the author of the once-anonymous blog Washingtonienne [60] we'd know that she is is almost certainly a Washington, D.C. resident.

the anonymous blog, confirming a match. The purpose of the adversary's tools is thus to narrow the field of possible authors of the anonymous text enough that another approach to identifying the author becomes feasible. As a result, in our experiments we test classifiers which can estimate the probability of a label applying to a sample, rather than those which can only return the most likely label. It is also worth noting that, in this scenario, there is no hard and fast distinction between classification and clustering. None of the blogs may be labeled with a name, but if enough material is linked together, it may be possible to track down the author.

As in so many other research projects, the main challenge in designing our experiments is the absence of a large dataset labeled with ground truth. To measure the feasibility of matching one blog to another from the same author, we of course need a set of blogs already grouped by author. If the experiments are to reflect matching at anything resembling the scale of the Internet, manually collecting enough examples will be impossible.

As a result, our first approach to conducting experiments is to simulate the case of an individual publishing two blogs by dividing the posts of a single blog into two groups; we then measure our ability to match the two groups of posts back together. Specifically, we select one of a large number of blogs and set aside several of its posts for testing. These test posts represent the anonymously authored content, while the other posts of that blog represent additional material from the same author found elsewhere on the Internet. We next train a classifier to recognize the writing style of each of the blogs in the entire dataset, taking care to exclude the test posts when training on the blog from which they were taken. After completing the training, we present the test posts to the classifier and use it to rank all the blogs according to their estimated likelihood of producing the test posts. If the source of the test posts appears near the top of the resulting list of blogs, the writing style of its author may be considered especially identifiable. As will be shown in Section 5.5, our experiences applying this process have revealed surprisingly high levels of identifiability: using only three test posts, the correct blog is ranked first out of 100,000 in over eight percent of trials.

At this point, the astute reader is no doubt brimming with objections to the methodology described above—and rightly so. How can we be sure that any linking we detect in this way is unintentional? Suppose a blog author signs each of their posts by adding their name at the end. They would not be at all surprised to discover that an automated tool can determine that the posts have the same author. More subtly, rather than linking posts based on similarities in writing style, our classifiers may end up relying on similarities in subject matter, such as specific words related to the topic of the blog. We take the following strategies in avoiding these pitfalls:

1. We begin by filtering out any obvious signatures in the posts. We also remove markup and any other text that does not appear to be directly entered by a human in order to avoid linking based on the blog software used.
2. We carefully limit the features we extract from each post and provide to the classifier. In particular, unlike previous work on author identification, we do not employ a "bag of

words" or any other features that can discover and incorporate arbitrary content. Our word-based features are limited to a fixed set of function words which bear little relation to the subject of discussion (e.g., "the," "in," etc.). While we do make use of single character frequencies, we exclude bigrams and trigrams, which may be significantly influenced by specific words.

3. We follow up the experiments described above (post-to-blog matching) with additional experiments which actually involve matching distinct blogs to one another. Specifically, we assembled a small collection of sets of blogs with the same author; for these experiments (blog-to-blog matching), we set aside one blog as the test content, mix the others from the same author into the full dataset of 100,000 blogs, and then measure our ability to pick them back out.

The results of the blog-to-blog matching experiments closely match the post-to-blog matching results, and we also found the results were not dominated by any one class of features. These facts have given us confidence that our methods are in fact discovering links in writing *style*—not blog software or the topic of a blog.

5.3 Data Sources and Features

Having given some high-level motivation for our experimental approach and methodology, we now detail our sources of data, the steps we took to filter it, and the feature set implemented.

Data sources. The bulk of our data was obtained from the ICWSM 2009 Spinn3r Blog Dataset, a large collection of blog posts made available to researchers by Spinn3r.com, a provider of blog-related commercial data feeds [23]. For the blog-to-blog matching experiments, we supplemented this by scanning a dataset of 3.5 million Google profile pages for users who specify multiple URLs [75]. Most of these URLs link to social network profiles rather than blogs, so we further searched for those containing terms such as "blog," "journal," etc. From this list of URLs, we obtained RSS feeds and individual blog posts.

We passed both sets of posts through the following filtering steps. First, we removed all HTML and any other markup or software-related debris we could find, leaving only (apparently) manually entered text. Next, we retained only those blogs with at least 7,500 characters of text across all their posts, or roughly eight paragraphs. Non-English language blogs were removed using the requirement that at least 15% of the words present must be among the top 50 English words, a heuristic found to work well in practice. Of course, our methods could be applied to almost any other language, but some modifications to the feature set would be necessary. To avoid matching blog posts together based on a signature the author included, we removed any prefix or suffix found to be shared among at least three-fourths of the posts of a blog. Duplicated posts were also removed.

At the end of this process, 5,729 blogs from 3,628 Google profiles remained, to which we added 94,271 blogs from the Spinn3r dataset to bring the total to 100,000. Of the 3,628

Category	Description	Count
Length	number of words/characters in post	2
Vocabulary richness	Yule's K and frequency of *hapax legomena*, *dis legomena*, etc.	11
Word shape	frequency of words with all uppercase letters, all lowercase letters, etc.	5
Word length	frequency of words that have 1–20 characters	20
Letters	frequency of a to z, ignoring case	26
Digits	frequency of 0 to 9	10
Punctuation	frequency of `.?!,;:()"-'`	11
Special characters	frequency of `‘~@#$%^&*_+=[]{}|/<>`	21
Function words	frequency of words like "the," "of," and "then"	293
Syntactic category pairs	frequency of every pair (A, B), where A is the parent of B in the parse tree	789

Table 5.1: The features used for classification. Most take the form of frequencies, and all are real-valued.

retained Google profiles, 1,763 listed a single blog, 1,663 listed a pair of blogs, and the other 202 listed three to five. Our final dataset contained 2,443,808 blog posts, an average of 24 posts per blog (the median was also 24). Each post contained an average of 305 words, with a median of 335.

Features. From each blog post, we extracted 1,188 real-valued features, transforming the post into a high-dimensional vector. These feature vectors were the only input to our classifiers; the text of the blog post played no further role after feature extraction.

Table 5.1 summarizes the feature set. All but the last of these categories consist of features which reflect the distributions of words and characters in each of the posts and have been used in previous work on author identification [1]. The features in the second category are designed to reflect the size of the author's vocabulary by recording usages of rare words. A *hapax legomenon* is a word that is used exactly once in some text (a post in our case). We also record the frequency of words that appear twice (*dis legomena*), three times, and so on, producing ten features. In addition, we include Yule's K statistic, which aggregates all such values (not only the first ten) into a single measure of vocabulary richness [86]. The next category concerns capitalization of words, as we expect the level of adherence to capitalization conventions to act as a distinguishing component of an author's writing style given the unedited, free-form nature of written content on the Internet. In addition to the frequency of all uppercase and all lowercase words, we record the frequencies of those with only the first letter uppercase, those with an initial uppercase letter followed by a

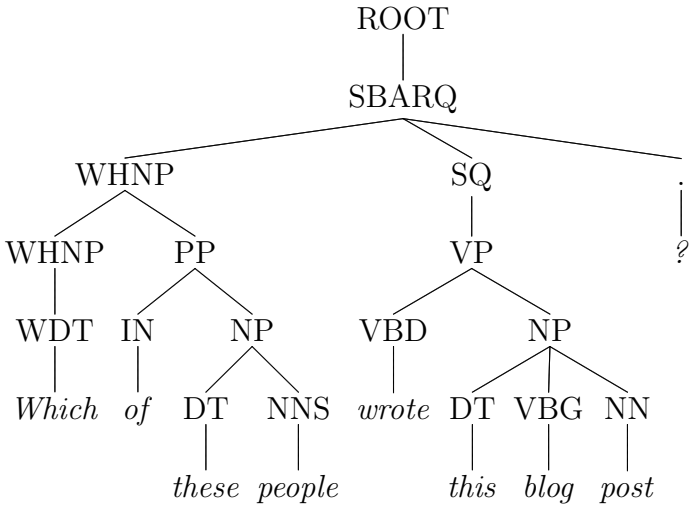


Figure 5.1: A sample parse tree produced by the Stanford Parser.

mix of upper and lowercase letters (camel case), and those with an initial lowercase letter followed by at least one uppercase letter (the only other possibility). We compute each of the letter frequency features as the number of occurrences of a specific letter in a post divided by the length of the post in characters. Other single-character frequencies are computed likewise, and word-frequency features are computed analogously, but at the level of words. Appendix D includes the full list of function words used for the second-to-last category.

For the last category of features in Table 5.1, we use the Stanford Parser [56] to determine the syntactic structure of each of the sentences in the input posts. As output, it produces a tree for each sentence where the leaf nodes are words and punctuation used, and other nodes represent various types of syntactic categories (phrases and parts of speech). Figure 5.1 shows an example parse tree as produced by the Stanford Parser, with tags such as NN for noun, NP for noun phrase, and PP for prepositional phrase. We generate features from the parse trees by taking each pair of syntactic categories that can appear as parent and child nodes in a parse tree tree, and counting the frequency of each such pair in the input data. Only immediate parent-child relationships are counted, so in the case of Figure 5.1, (SQ,VP) would be recorded but (SQ,NP) would not.

To gain a better intuitive understanding of the relative utility of the features and for use in feature selection, we computed the information gain of each feature over the entire dataset [42]. We define information gain as

$$IG(F_i) = H(B) - H(B|F_i) = H(B) + H(F_i) - H(B, F_i),$$

where H denotes Shannon entropy, B is the random variable corresponding to the blog number, and F_i is the random variable corresponding to feature i. Since the features are

Feature	Information Gain in Bits
Frequency of ’	1.097
Number of characters	1.077
Freq. of words with only first letter uppercase	1.073
Number of words	1.060
Frequency of (NP, PRP) (noun phrase containing a personal pronoun)	1.032
Frequency of .	1.022
Frequency of all lowercase words	1.018
Frequency of (NP, NNP) (noun phrase containing a singular proper noun)	1.009
Frequency of all uppercase words	0.991
Frequency of ,	0.947

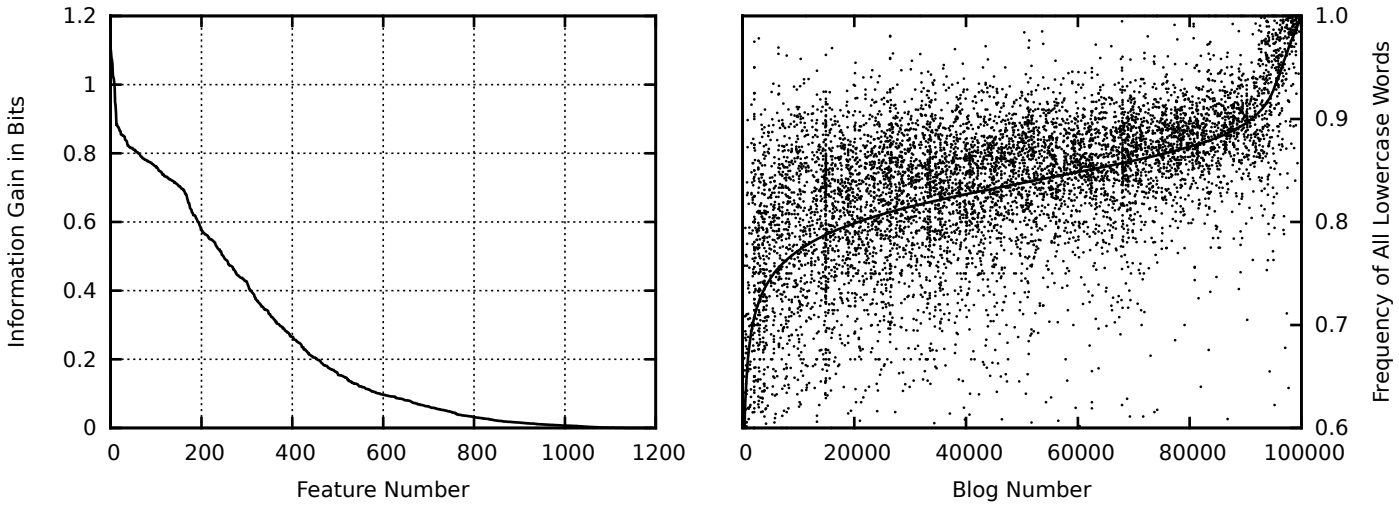
Table 5.2: The top ten features by information gain.

real-valued, and entropy is defined only for discrete random variables,[2] we need to sensibly map them to a set of discrete values. For each feature, we partitioned the range of observed values into twenty intervals. We reserved one bin for the value zero, given the sparsity of our feature set; the other nineteen bins were selected to contain approximately equal numbers of values across all the posts in the dataset.

A portion of the result of this analysis is given in Table 5.2, which lists the ten features with greatest information gain when computed as described; an extended version listing the top thirty appears in Appendix D. Several other binning methods were found to produce similar results. Base-two logarithms were used in the entropy computation, so the values given correspond to quantities of bits. With information gains ranging from 1.097 to 0.947, these features can all be considered roughly equal in utility. Perhaps least surprisingly, the length of posts (in words and characters) is among the best indicators of the blog the posts were taken from. Several punctuation marks also appear in the top ten, along with the three most common patterns of upper and lowercase letters and two syntactic category pairs. The information gains of all 1,188 features are shown in Figure 5.2 (a).

To give a sense of the typical variation of feature values both within a blog and between different blogs, Figure 5.2 (b) displays a representative example of one of the ten features in Table 5.2: the frequency of all lowercase words. The plot was generated by sorting the 100,000 blogs according to the mean of this feature across their posts. The means are shown by the solid line, and the values for individual posts are plotted as dots. For legibility, the figure only shows every third post of every one hundredth blog. As one might expect, the values

[2] A definition also exists for continuous random variables, but applying it requires assuming a particular probability mass function.



(a) The information gain of all features.

(b) Per-post values (dots) and per-blog means (line) of an example feature across the dataset.

Figure 5.2: The information gain of each feature and sample values for one feature.

vary from post to post by much larger amounts than the differences in mean between most pairs of blogs, indicating that this feature alone carries a fairly small amount of information. The corresponding plots for features with lower information gain look similar, but with less variation in means or more variation between individual posts from the same author. The two plots in Figure 5.2 make it clear that many features will need to be considered in combination if we are to effectively identify authors.

5.4 Classifiers

To conduct the two types of experiments described in Section 5.2, we used three types of classifiers: nearest neighbor, naive Bayes, and support vector machine. The first two of these are particularly straightforward to implement, and we used a freely available tool for the third [51], indicating that results similar to ours could be obtained by even an unsophisticated adversary. In describing each of these algorithms, we denote a labeled example as a pair $(\vec{x}, y)$, where $\vec{x} \in \mathbb{R}^n$ is a vector of features and $y \in \{1, 2, \ldots, m\}$ is the label. In our case, $n = 1,188$, the number of features; and $m = 100,000$, the number of blogs in our dataset. After training a classifier on the labeled examples, we present it with one or more unlabeled examples $\vec{x}_1, \vec{x}_2, \ldots$, test posts taken from a single blog in the case of post-to-blog matching experiments, or an entire blog in the case of blog-to-blog matching. In either case, we rank

the labels $\{1, 2, \ldots, m\}$ according to our estimate of the likelihood that the corresponding author wrote the unlabeled posts. One key difference between both of our experiments and the typical classification scenario is that we know that $\vec{x}_1, \vec{x}_2, \ldots$ have the same label.

Nearest neighbor. In the case of our nearest neighbor classifier, we begin by preprocessing the data in two ways that help to improve performance. First, we rescale each feature so that its mean over the entire training set is one; this is accomplished by dividing each feature value in each post by the mean of that feature. After doing so, for each post, we scale its vector of features to have Euclidean norm one. The training phase then proceeds by averaging the values of each feature over all the posts of a blog, forming a vector $(\mu_1, \ldots, \mu_n)$ of feature means that serves as a "fingerprint" for the blog. Like the vectors for the original posts, we also normalize each of these fingerprints so it has length one.

When given a series of unlabeled posts $\vec{x}_1, \vec{x}_2, \ldots$ for classification, we again average the value of each feature over the available posts, producing a fingerprint for the posts to be classified. This fingerprint is also normalized to have length one. Finally, we compute the Euclidean distance between the fingerprint of the posts and the fingerprint of each blog in turn and sort the blogs by distance to produce our ranking.

Naive Bayes. Our naive Bayes classifier is similar to the nearest neighbor classifier, but takes into account the variance of each feature in addition to its mean. Specifically, the training phase for each blog consists of computing the mean μ_i and variance σ_i^2 of each feature i over that blog's posts. A small-sample correction of 5×10^{-6} is added to each variance to prevent any from being exactly zero.[3] Then given an unlabeled post $\vec{x} = (x_1, \ldots, x_n)$, we assign a blog with means $(\mu_1, \ldots, \mu_n)$ and variances $(\sigma_1^2, \ldots, \sigma_n^2)$ the score

$$\sum_{i=1}^{n} -\log(\sigma_i^2) - \frac{(x_i - \mu_i)^2}{\sigma_i^2}$$

and rank the blogs according to their score, with greater (i.e., less negative) scores indicating that authorship is more likely. If more than one unlabeled post is available, we compute each score as above, but summing over the features in all the unlabeled posts. Frequently arising in naive Bayes classifiers, the above expression is obtained by fitting a Gaussian to each feature and assuming the features are independent given the label (the naive assumption). Since we only aim to rank the labels by their probability of producing a sample, we take the logarithm of each probability to simplify the calculations, then remove terms that affect each probability equally, resulting in the "score" computed for each blog. Our naive Bayes classifier was found

[3]This will occur, for example, whenever one particular function word is not used in any of the posts of a blog that appear in the training set. If a small-sample correction were not used and that word appeared in an unlabeled post, the zero variance would prevent that blog from being selected no matter what other evidence was present.

to perform best when given only the 400 features with greatest information gain; the other two classifiers used the entire feature set.

Support vector machine. While support vector machines were originally formulated for binary classification [12, 35], several methods exist for applying them to classification problems with more than two labels. Due to the very large number of labels in our case, we employ the one-versus-all strategy for reducing the problem to training a series of binary SVMs, one for each label. During training, each SVM is given all posts of the corresponding blog that are in the training set as positive examples and a selection of posts from other blogs as negative examples. Ideally, we would provide each SVM with all posts from other blogs as negative examples. Since the large size of our dataset makes this infeasible, we instead use the posts from a sample of 1,000 other blogs as negative examples. To improve the ability of an SVM to distinguish between its associated blog and other, similar blogs, we ensure that this set of 1,000 blogs includes the 100 that are "nearest" to it, using the same notion of Euclidean distance between fingerprints that we explained in the description of the nearest neighbor classifier. The remaining 900 are selected uniformly at random.

To classify a single unlabeled post, we present it to each SVM and record the resulting margins between the post and each of the separating hyperplanes. When we instead have a group of unlabeled posts by the same author, we sum the margins resulting from classifying each post separately. To produce the ranked list of authors for the post(s), we sort the labels by the margins we obtained from the corresponding SVMs. Larger (more positive) margins indicate a higher likelihood that the corresponding label applies to the new post(s) while smaller (more negative) margins indicates a lower likelihood.

As with our nearest neighbor classifier, we rescaled each feature such that its mean would be one and normalized each post to be of length one before training the SVMs. No kernel function was used.

5.5 Experimental Results

In Figures 5.3 and 5.4, which summarize our most important results, we provide the full distribution of outcomes obtained by applying the three classifiers to the two types of experiments introduced in Section 5.2. We now give the details of the procedure used to obtain these results.

In each trial of the first experiment, post-to-blog matching, we randomly selected three posts of one blog and set them aside as the testing data. The three classifiers were then used to rank each blog according to its estimated likelihood of producing the test posts. Of course, we were careful to ensure that the classifiers were not given the test posts during training. For this experiment, we only selected blogs from the Spinn3r dataset as the source of test posts, but we used the classifiers to rank all 100,000 blogs. In each trial, we recorded the rank of the correct blog; Figure 5.3 displays the CDF of these rankings.

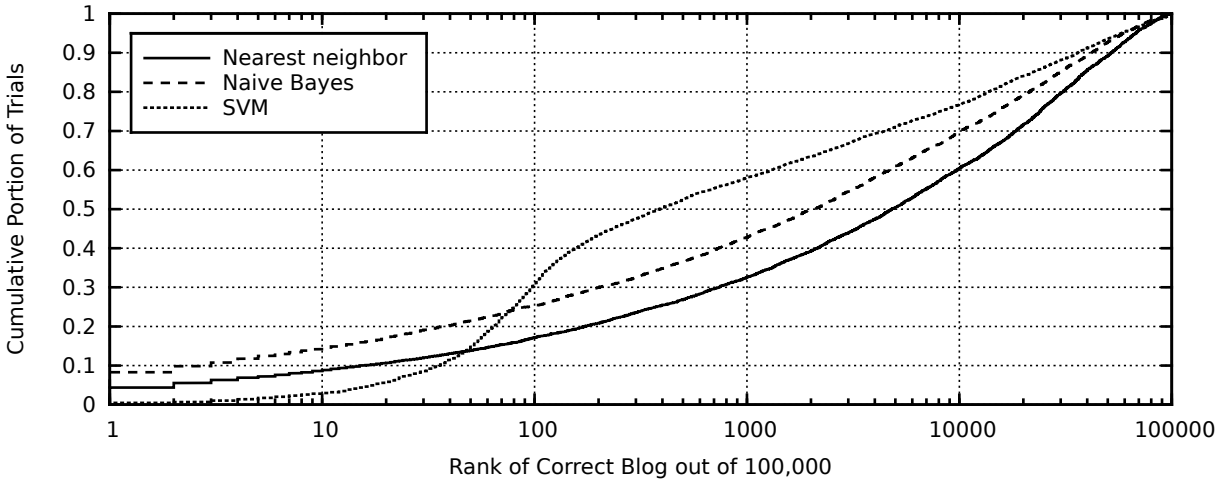


Figure 5.3: Results of the post-to-blog matching experiments, using three posts (roughly 900 words).

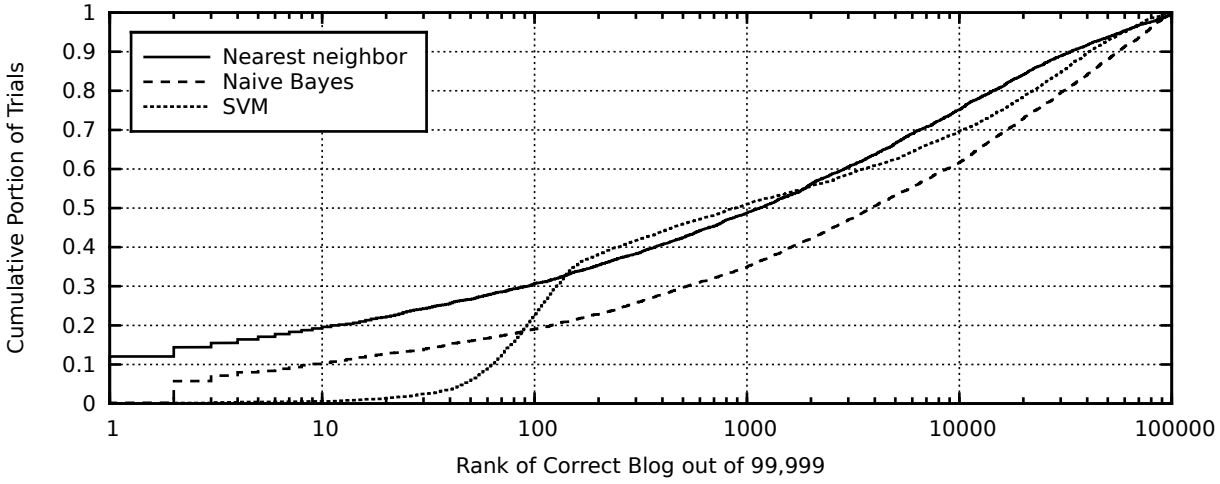


Figure 5.4: Results of the blog-to-blog matching experiments.

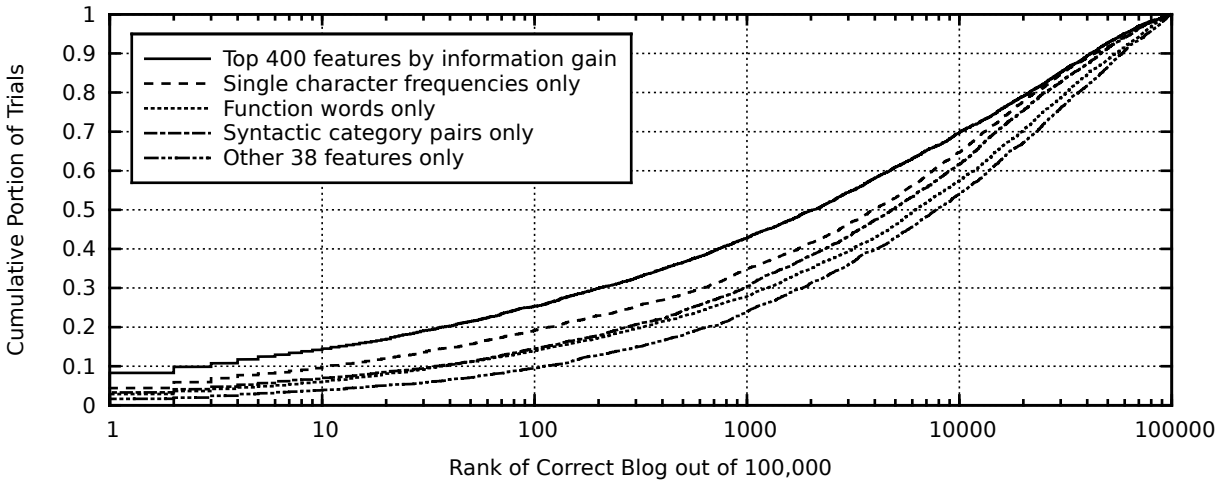


Figure 5.5: Post-to-blog matching using three posts and naive Bayes while limiting the feature set. The full feature set used in the other experiments is shown by the solid line for comparison.

Each trial of the blog-to-blog matching experiment consisted of randomly selecting one of the blogs obtained from the Google profiles, then ranking the other 99,999 blogs based on their similarity to its posts. The rank of the other blog listed on the same Google profile was then saved as the outcome, producing the results given in Figure 5.4. In the (uncommon) case of more than one additional blog listed on the Google profile, the highest ranked was considered the outcome, on the grounds that an adversary would be interested in linking an anonymous blog to any material from the same author.

Several interesting results can be directly read off the graphs. Tracing up from the x-axis in Figure 5.3, we see that the naive Bayes classifier ranked the correct blog first in approximately 8% of trials (its classification accuracy) and within the top ten in about 15% of trials. In the case of blog-to-blog matching, the nearest neighbor classifier ranked a blog from the correct author first and within the top ten in approximately 12% and 20% of trials, respectively. No one algorithm was clearly superior to the other two. While the naive Bayes and nearest neighbor classifiers achieved higher accuracies, the SVM classifier produced the best median outcomes. This can be seen by tracing right from the y-axis, which reveals median rankings of about 400 and 900 in the two experiments. With a one-half chance of finding the correct blog both above and below these rankings, they can be considered the most typical outcomes. Considering the fact that we are applying methods not (to our knowledge) previously used on more than 300 authors, we find these results for 100,000 authors to be surprisingly good.

To help confirm the validity of these results, we manually inspected a small sample of the blogs that were most easily matched in each experiment, since these would be the ones most

likely to contain any post signatures or other illegitimate text that might have escaped our filtering. Nothing that could significantly affect the results was found. As a further check, we reran the experiments with the naive Bayes classifier while using only subsets of our features, in order to determine whether one particular type of feature was especially crucial to its performance. The results for post-to-blog matching are shown in Figure 5.5, along with the original naive Bayes results[4] from Figure 5.3 for comparison. While performance suffers with the limited feature sets, it is clear that no one type of feature is strictly necessary for effective matching. For example, with the full feature set, the correct blog is in the top 100 in 25% of trials, but only 38 features are necessary to achieve the same result in 10% of trials. The case of blog-to-blog matching with the limited feature sets is similar: performance is reduced, but not overwhelmingly.

The next two graphs shown help elucidate the relationship between the identifiability of an author and the amounts of labeled and unlabeled content available. Figure 5.6 displays the result of repeating the post-to-blog matching experiment with only one unlabeled post rather than three. At an average of 305 words, one post is similar in size to a comment on a news article or a message board post, so these results are indicative of the ability to match an isolated online posting against our blog corpus. While the identifiability of such small amounts of text is markedly reduced, we still achieve approximately 4% accuracy with the naive Bayes classifier and a median rank of about 2,500 with the SVM classifier.

Referring back to Figure 5.3, the significant difference between the tenth percentile and median outcomes (for naive Bayes, rankings of 2 and about 2,000 respectively) suggests that some blogs may be considerably more identifiable than others. Unsurprisingly, much of this difference is explained by the size of the blog and resulting amount of labeled content, as shown in Figure 5.7. The largest blogs, with at least 16,000 words (roughly 50 posts) are ranked within the top 100 out of 100,000 by naive Bayes in 60% of trials, and ranked first in about 27%. On the other hand, for blogs of less than 2,000 words (excluding the three test posts), those numbers are reduced to about 23% and 3%. This suggests that authors who wish to publish anonymously should consider the amount of material they have already written that appears online.

[4]Recall that we found this classifier to perform best with the 400 features with greatest information gain rather than all 1,188.

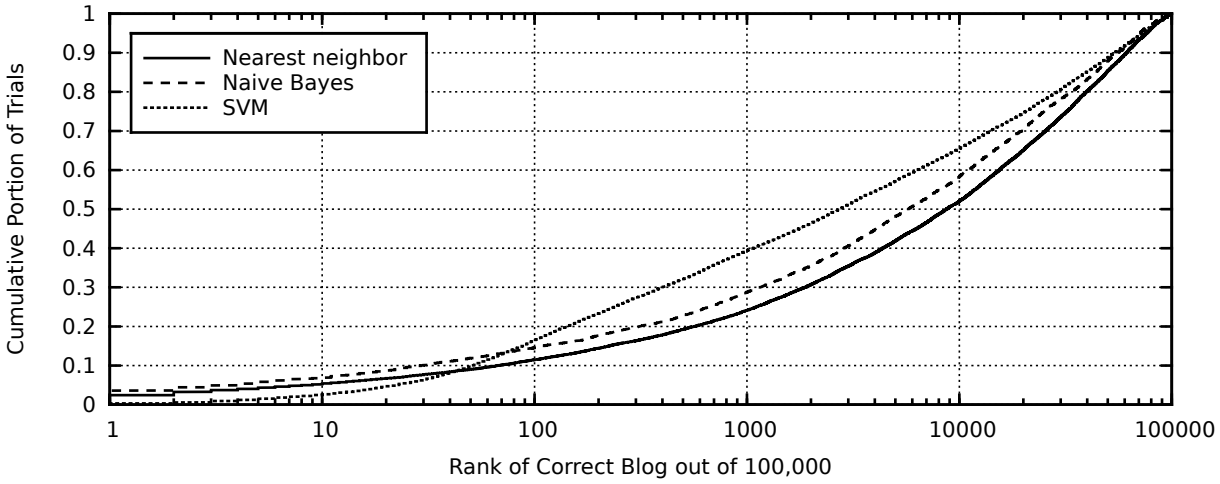


Figure 5.6: Post-to-blog matching using only one post (roughly 300 words).

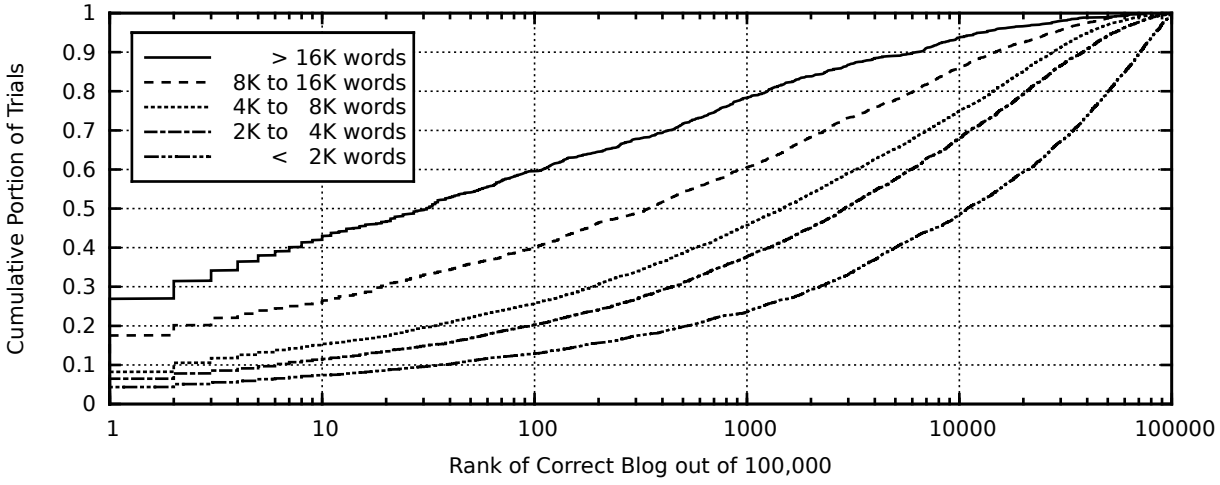


Figure 5.7: Post-to-blog matching results using three posts and naive Bayes, by amounts of labeled content.

Chapter 6

Conclusions and Future Work

In this work, we've presented a collection of new cryptographic tools aimed at giving individuals more control over their identity. Through our proposal for signatures of reputation, we hope to enable future applications which allow anonymous, yet credible publication of information, and our scheme for private stream searching is intended to enable retrieval of information while placing similar protections on the user's privacy. Many problems remain for future research. While feasible with current technology, both schemes are too costly to be used in situations where they provide only a minor benefit; more efficient solutions would be more broadly applicable. Our signatures of reputation support only monotonic reputation, but in many applications, "bad" reputation is the most important kind. Finding a way to support negative feedback will require innovative definitions as well as cryptographic constructions. Other challenges include devising a scheme which maintains privacy despite a malicious registration authority or, more ambitiously, replaces it with an entirely decentralized approach to limiting the Sybil attack.

The final area of research presented in this document suggests fundamental limitations to anonymous speech, regardless of the aforementioned methods. If *what* is said is intrinsically linked to the person who said it, no cryptographic tools can give the ability to speak both publicly and anonymously. Our findings have shown that individuals who have already authored large amounts of publicly available text may no longer be able to publish anonymously. Fortunately, it is not yet clear that a significant risk of identification through writing style exists for the broader population. However, our results can only be interpreted as a lower bound on the severity of this risk, which is likely to increase over time.

Bibliography

[1] Ahmed Abbasi and Hsinchun Chen. Writeprints: A stylometric approach to identity-level identification and similarity detection in cyberspace. *ACM Transactions on Information Systems*, 26(2), March 2008.

[2] Elli Androulaki, Seung Geol Choi, Steven M. Bellovin, and Tal Malkin. Reputation systems for anonymous networks. In *Privacy Enhancing Technologies*, 2008.

[3] Michael Arrington. AOL proudly releases massive amounts of user search data. *TechCrunch News*, August 2006.

[4] Man Ho Au, Willy Susilo, and Yi Mu. Practical anonymous divisible e-cash from bounded accumulators. In *Financial Cryptography*, 2008.

[5] Man Ho Au, Q. Wu, Willy Susilo, and Yi Mu. Compact e-cash from bounded accumulator. In *CT-RSA*, 2007.

[6] Eric Backer and Peter van Kranenburg. On musical stylometry–a pattern recognition approach. *Pattern Recognition Letters*, 26(3):299 – 309, 2005.

[7] Lars Backstrom, Cynthia Dwork, and Jon M. Kleinberg. Wherefore art thou r3579x? In *International World Wide Web Conference*, 2007.

[8] Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. Randomizable proofs and delegatable anonymous credentials. In *Crypto*, 2009.

[9] Mira Belenkiy, Melissa Chase, Chris Erway, John Jannotti, Alptekin Kupcu, Anna Lysyanskaya, and Eric Rachlin. Making p2p accountable without losing privacy. In *WPES*, 2007.

[10] Mira Belenkiy, Melissa Chase, Markulf Kohlweiss, and Anna Lysyanskaya. Non-interactive anonymous credentials. In *TCC*, 2008.

[11] Mihir Bellare, Alexandra Boldyreva, Anand Desai, and David Pointcheval. Key-privacy in public-key encryption. In *Asiacrypt*, 2001.

[12] Isabelle Guyon Bernhard E. Boser and Vladimir Vapnik. A training algorithm for optimal margin classifiers. In *COLT*, pages 144–152, 1992.

[13] Burton Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[14] James Bone. Vogue model Liskula Cohen wins right to unmask offensive blogger. *The Times*, August 2009.

[15] Dan Boneh and Xavier Boyen. Short signatures without random oracles and the SDH assumption in bilinear groups. *Journal of Cryptology*, 21(2), 2008.

[16] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *Crypto*, 2004.

[17] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Guiseppe Persiano. Public key encryption with keyword search. In *Eurocrypt*, 2004.

[18] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF formulas on ciphertexts. In *Theory of Cryptography Conference*, 2005.

[19] Joseph K. Bradley, Patrick Gage Kelley, and Aaron Roth. Author identification from citations. Technical report, Carnegie Mellon University, December 2008.

[20] Michael Brennan and Rachel Greenstadt. Practical attacks against authorship recognition techniques. In *IAAI*, 2009.

[21] Andrei Broder and Michael Mitzenmacher. Network applications of Bloom filters: A survey. *Internet Mathematics*, 1(4):485–509, 2005.

[22] Jeff Brumley. Unmasked blogger blames First Baptist, sheriff's office. *The Florida Times-Union*, April 2009.

[23] K. Burton, A. Java, and I. Soboroff. The ICWSM 2009 Spinn3r dataset. In *International AAAI conference on weblogs and social media*, 2009.

[24] Christian Cachin, Silvio Micali, and Markus Stadler. Computationally private information retrieval with polylogarithmic communication. In *Eurocrypt*, 1999.

[25] Jan Camenisch, Susan Hohenberger, Markulf Kohlweiss, Anna Lysyanskaya, and Mira Meyerovich. How to win the clone wars: Efficient periodic n-times anonymous authentication. In *CCS*, 2006.

[26] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact e-cash. In *Eurocrypt*, 2005.

[27] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Balancing accountability and privacy using e-cash. In *Security and Cryptography for Networks (SCN)*, 2006.

[28] Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. An accumulator based on bilinear maps and efficient revocation for anonymous credentials. In *PKC*, 2009.

[29] Jan Camenisch, Anna Lysyanskaya, and Mira Meyerovich. Endorsed e-cash. In *IEEE Symposium on Security and Privacy*, 2007.

[30] Jan Camenisch and Markus Stadler. Proof systems for general statements about discrete logarithms. Technical Report 260, Inst. for TCS, ETH Zurich, 1997.

[31] Yan-Cheng Chang. Single database private information retrieval with logarithmic communication. In *Information Security and Privacy (ACISP)*, 2004.

[32] Benny Chor, Niv Gilboa, and Moni Naor. Private information retrieval by keywords. Technical Report CS0917, Department of Computer Science, Technion, 1997.

[33] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *Foundations of Computer Science (FOCS)*, 1995.

[34] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9:251–280, 1990.

[35] Corinna Cortes and Vladimir Vapnik. Support-vector networks. In *Machine Learning*, volume 20, pages 273–297, 1995.

[36] Ingemar Cox, Matthew Miller, and Jeffery Bloom. *Digital Watermarking*. Morgan Kaufmann, 2002.

[37] Ivan Damgard, Jens Groth, and Gorm Salomonsen. The theory and implementation of an electronic voting system. In *Secure Electronic Voting*, 2003.

[38] Ivan Damgård and Mads Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In *Public Key Cryptography (PKC)*, 2001.

[39] Olivier Y. de Vel, Alison Anderson, Malcolm Corney, and George M. Mohay. Mining email content for author identification forensics. *SIGMOD Record*, 30(4):55–64, 2001.

[40] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *USENIX Security Symposium*, 2004.

[41] John R. Douceur. The Sybil attack. In *International Workshop on Peer-to-Peer Systems*, 2002.

[42] George Forman. An extensive empirical study of feature selection metrics for text classification. Technical report, Hewlett-Packard Labs, 2002.

[43] Freedman, Ishai, Pinkas, and Reingold. Keyword search and oblivious pseudorandom functions. In *Theory of Cryptography Conference (TCC)*, 2005.

[44] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *Eurocrypt*, 1999.

[45] Eu-Jin Goh. Secure indexes. Cryptology ePrint Archive, Report 2003/216, 2003.

[46] Jens Groth. Evaluating security of voting schemes in the universal composability framework. In *ACNS*, 2004.

[47] Jens Groth. Non-interactive zero-knowledge arguments for voting. In *ACNS*, 2005.

[48] Jens Groth. Fully anonymous group signatures without random oracles. In *Asiacrypt*, 2007.

[49] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In *Eurocrypt*, 2008.

[50] Shawndra Hill and Foster J. Provost. The myth of the double-blind review? Author identification using only citations. *SIGKDD Explorations*, 5(2):179–184, 2003.

[51] Thorsten Joachims. Making large-scale SVM learning practical. In *Advances in Kernel Methods—Support Vector Learning*, 1999.

[52] Patrick Juola. Authorship attribution. *Foundations and Trends in Information Retrieval*, 1:233–334, December 2006.

[53] Gary Kacmarcik and Michael Gamon. Obfuscating document stylometry to preserve author anonymity. In *Meeting of the Association for Computational Linguistics*, 2006.

[54] J. Kahn, J. Komlós, and E. Szemerédi. On the probability that a random ± 1 matrix is singular. *Journal of the American Mathematical Society*, 8(1):223–240, 1995.

[55] Eike Kiltz. Chosen-ciphertext security from tag-based encryption. In *TCC*, 2006.

[56] Dan Klein and Christopher D. Manning. Accurate unlexicalized parsing. In *Meeting of the Association for Computational Linguistics*, pages 423–430, 2003.

[57] J. Komlós. On the determinant of (0,1)-matrices. *Studia Math. Hungarica*, 2:7–21, 1967.

[58] Kaoru Kurosawa and Wakaha Ogata. Oblivious keyword search. *Journal of Complexity*, 20, 2004.

[59] Eyal Kushilevitz and Rafail Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *Foundations of Computer Science (FOCS)*, 1997.

[60] Richard Leiby. The Hill's sex diarist reveals all (well, some). *The Washington Post*, May 2004.

[61] Wei-Jen Li, Ke Wang, S J Stolfo, and B Herzog. Fileprints: identifying file types by n-gram analysis. In *IEEE Systems, Man, and Cybernetics Information Assurance Workshop*, 2005.

[62] Helger Lipmaa. An oblivious transfer protocol with log-squared communication. In *Information Security Conference (ISC)*, 2005.

[63] Ben Lynn. The Pairing-Based Cryptography (PBC) library. `http://crypto.stanford.edu/pbc`.

[64] Philip MacKenzie, Michael Reiter, and Ke Yand. Alternatives to non-malleability: definitions, constructions, and applications. In *TCC*, 2004.

[65] David Madigan, Alexander Genkin, David D. Lewis, Shlomo Argamon, Dmitriy Fradkin, and Li Ye. Author identification on the large scale. In *Joint Meeting of the Interface and Classification Society of North America*, 2005.

[66] H. Maurer, F. Kappe, and B. Zaka. Plagiarism—a survey. *Journal of Universal Computer Science*, 12(8):1050–1084, 2006.

[67] Ralph C. Merkle. A certified digital signature. In *Crypto*, 1989.

[68] A. Miyaji, M. Nakabayashi, and S. Takano. New explicit conditions of elliptic curve traces for FR-reduction. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 84(5):1234–1243, 2001.

[69] F Mosteller and D L Wallace. *Inference and Disputed Authorship: The Federalist*. Addison-Wesley, 1964.

[70] Moni Naor and Benny Pinkas. Oblivious transfer and polynomial evaluation. In *Symposium on Theory of Computing (STOC)*, 1999.

[71] Arvind Narayanan and Vitaly Shmatikov. Robust de-anonymization of large sparse datasets. In *IEEE Symposium on Security and Privacy*, 2008.

[72] Rafail Ostrovsky and William Skeith. Private searching on streaming data. In *Crypto*, 2005.

[73] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank citation ranking: bringing order to the web. Technical Report 1999-66, Stanford InfoLab, 1999.

[74] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Eurocrypt*, 1999.

[75] Daniele Perito, Claude Castelluccia, Mohamed Ali Kaafar, and Pere Manils. How unique and traceable are usernames? `http://arxiv.org/abs/1101.5578v3`, 2011.

[76] Franziska Pingel and Sandra Steinbrecher. Multilateral secure cross-community reputation systems for internet communities. In *TrustBus*, 2008.

[77] Maxim Shevertalov, Jay Kothari, Edward Stehle, and Spiros Mancoridis. On the use of discretized source code metrics for author identification. In *International Symposium on Search Based Software Engineering*, 2009.

[78] Victor Shoup. Lower bounds for discrete logarithms and related problems. In *Eurocrypt*, 1997.

[79] Robert Silverman. A cost-based security analysis of symmetric and asymmetric key lengths. Technical report, RSA Laboratories, November 2001.

[80] Dawn Xiaodong Song, David Wagner, and Adriane Perrig. Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy*, 2000.

[81] Sandra Steinbrecher. Design options for privacy-respecting reputation systems within centralised internet communities. In *International Information Security Conference (SEC)*, 2006.

[82] Sandra Steinbrecher. Enhancing multilateral security in and by reputation systems. In *FIDIS/IFIP Internet Security and Privacy Summer School*, September 2008.

[83] Terence Tao and Van H. Vu. On random ± 1 matrices: singularity and determinant. In *Symposium on Theory of Computing (STOC)*, pages 431–440, 2005.

[84] Terence Tao and Van H. Vu. On the singularity probability of random bernoulli matrices. *Journal of the American Mathematical Society*, 20(3):603–628, 2007.

[85] Community forum policy, online edition of the Wall Street Journal. Retrieved from `http://online.wsj.com/` on April 25, 2010.

[86] G. Udny Yule. *The Statistical Study of Literary Vocabulary.* Cambridge University Press, 1944.

Appendix A

The Hardness of SCDH in Generic Groups

The SCDH ("stronger than CDH") assumption may be stated as follows.

Let $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a bilinear map, where the groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are of prime order p. Let $\varphi : \mathbb{G}_2 \to \mathbb{G}_1$ be an efficiently computable isomorphism. Assume g_2 is a generator of $\mathbb{G}_2$ and let $g_1 = \varphi(g_2)$. Let $\rho \in \mathbb{Z}_p^*$. Select $r, s \xleftarrow{R} \mathbb{Z}_p \setminus \{-\rho\}$, $h \xleftarrow{R} \mathbb{G}_1$, $u, v \xleftarrow{R} \mathbb{G}_2$. Then given

$$\rho, g_1, h, g_2, u, v, u^r, v^s, g_1^r, h^s, g_2^{\frac{1}{\rho+r}}, g_2^{\frac{1}{\rho+s}} \quad ,$$

it is computationally infeasible to output a triple $(z, z^r, z^s) \in \mathbb{G}_1^3$ where $z \neq 1$.

We prove this in the generic group model [78] by providing an upper bound on the probability that an adversary is able to output such a triple.

A.1 Generic Group Formulation of SCDH

In this model, we assume elements of $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ are identified only by random string identifiers. Specifically, we identify the elements of $\mathbb{G}_1$ using an injective map $\xi_1 : \mathbb{Z}_p \to \{0,1\}^k$ selected uniformly at random, where k is sufficiently large that we will assume the adversary cannot guess any valid element identifiers. For any $x \in \mathbb{Z}_p$, the identifier $\xi_1(x)$ represents the element $g_1^x \in \mathbb{G}_1$. The elements of $\mathbb{G}_2$ and $\mathbb{G}_T$ are similarly identified via maps ξ_2 and ξ_3. To select the random elements h, u, v, we select random exponents x, y_1, y_2 and let $h = g_1^x$, $u = g_2^{y_1}$, and $v = g_2^{y_2}$; in this way their identifiers may be computed using ξ_1 and ξ_2.

The adversary will start with the identifiers of the elements in the challenge and must query an oracle $\mathcal{O}$ to compute the group operation in any of the three groups, to evaluate the bilinear map, or to evaluate φ. Upon termination, it must output three strings $\pi, \pi', \pi'' \in \{0,1\}^k$. If there exists a $z \in \mathbb{Z}_p$ such that $\pi = \xi_1(z)$, $\pi' = \xi_1(zr)$, and $\pi'' = \xi_1(zs)$, then the adversary has won the game.

Theorem A.1.1. *For any adversary $\mathcal{A}$ making at most q queries to the oracle $\mathcal{O}$,*

$$\Pr\left[\begin{array}{c|c} \mathcal{A}^{\mathcal{O}}\left(\begin{array}{c} p, \rho, \xi_1(1), \xi_1(x), \xi_1(r), \xi_1(xs), \xi_2(1), \xi_2(y_1), \\ \xi_2(y_2), \xi_2(y_1 r), \xi_2(y_2 s), \xi_2\left(\frac{1}{r+\rho}\right), \xi_2\left(\frac{1}{s+\rho}\right) \end{array}\right) = \pi, \pi', \pi'' \ \wedge & x, y_1, y_2 \xleftarrow{R} \mathbb{Z}_p \\ \exists z \in \mathbb{Z}_p \text{ such that } \pi = \xi_1(z), \pi' = \xi_1(zr), \text{ and } \pi'' = \xi_1(zs) & r, s \xleftarrow{R} \mathbb{Z}_p \setminus \{-\rho\} \end{array}\right] \leq \frac{24(q+11)^2}{p} \ .$$

Our proof employs the following strategy. First, we define an alternative (the "formal game") to the real game described above. Next, we show that it is impossible for the adversary to win the formal game. Finally, we show that with probability at least $1 - \frac{24(q+11)^2}{p}$, the view of an adversary that played the formal game is identical to what their view would have been if they were playing the real game.

A.2 Formal Game

In this version of the game, the oracle $\mathcal{O}$ ignores the actual values of x, y_1, y_2, r, and s and instead treats them as formal variables X, Y_1, Y_2, R, and S (the exponent ρ is treated differently, as will be explained).

Specifically, $\mathcal{O}$ maintains three lists L_1, L_2, L_3 of pairs. In each pair (π, F), π is an identifier string and F is a rational function with indeterminates X, Y_1, Y_2, R, S. That is, F is a member of the field of rational functions $\mathbb{Z}_p(X, Y_1, Y_2, R, S)$. The elements of the field $\mathbb{Z}_p(X, Y_1, Y_2, R, S)$ may be considered to be (multivariate) polynomial fractions $\frac{P}{Q}$, where P and Q are in the polynomial ring $\mathbb{Z}_p[X, Y_1, Y_2, R, S]$. More precisely, F is an equivalence class of such fractions, where $\frac{P_1}{Q_1} = \frac{P_2}{Q_2}$ if $P_1 Q_2 = P_2 Q_1$. In the following, we identify a rational function F with the representative element of its equivalence class $\frac{P}{Q}$ that is written in lowest terms (i.e., $\gcd(P, Q) = 1$).

Each list is initialized with the identifiers of and the polynomial fractions corresponding to the challenge values in each of the three groups. So initially

$$\begin{aligned} L_1 &= ((\pi_{1,1}, 1), (\pi_{1,2}, X), (\pi_{1,3}, R), (\pi_{1,4}, XS)) \\ L_2 &= \left((\pi_{2,1}, 1), (\pi_{2,2}, Y_1), (\pi_{2,3}, Y_2), (\pi_{2,4}, Y_1 R), (\pi_{2,5}, Y_2 S), \left(\pi_{2,6}, \frac{1}{R+\rho}\right), \left(\pi_{2,7}, \frac{1}{S+\rho}\right)\right) \\ L_3 &= () \ . \end{aligned}$$

Note that only X, Y_1, Y_2, R, and S are indeterminates in the above polynomial fractions; ρ is simply a constant in $\mathbb{Z}_p$. Now whenever the adversary makes a query to perform the group operation in $\mathbb{G}_1$ on $\pi_{1,i}$ and $\pi_{1,j}$ (including a selection bit specifying whether they wish to multiply or divide), we look up in list L_1 the corresponding polynomial fractions $F_{1,i}, F_{1,j}$ and compute $F = F_{1,i} \pm F_{1,j}$. We check whether F (in a canonical form) already exists in L_1 and, if so, return the corresponding identifier. Otherwise, we randomly select a new identifier π and append (π, F) to L_1. Queries for the group operations in $\mathbb{G}_2$ and $\mathbb{G}_T$ are

answered analogously. To answer queries for the bilinear map, we compute $F = F_{1,i} \cdot F_{2,j}$ and check L_3 for F, again, adding it if it was not already present. To answer queries for the isomorphism φ applied to some $F_{2,i}$, we simply check L_1 for $F_{2,i}$.

We now argue that it is impossible for the adversary to win when $\mathcal{O}$ responds to queries using the rules of the formal game. In order to win, the adversary must construct polynomial fractions F_{1,i_1}, F_{1,i_2}, and F_{1,i_3} in L_1, where $F_{1,i_2} = F_{1,i_1} \cdot R$, $F_{1,i_3} = F_{1,i_1} \cdot S$, and $F_{1,i_1} \neq 0$. However, any $F_{1,*}$ which the adversary can construct in L_1 is of the form

$$F_{1,*} = a_1 + a_2X + a_3R + a_4XS + a_5Y_1 + a_6Y_2 + a_7Y_1R + a_8Y_2S + \frac{a_9}{R+\rho} + \frac{a_{10}}{S+\rho} \ ,$$

where $a_1, \ldots, a_{10} \in \mathbb{Z}_p$.[1] So if $F_{1,i_2} = F_{1,i_1} \cdot R$, then F_{1,i_1} must be of the form $F_{1,i_1} = a_3 + a_7Y_1$. Similarly, if $F_{1,i_3} = F_{1,i_1} \cdot S$, then F_{1,i_1} must be of the form $F_{1,i_1} = a_4X + a_8Y_2$.

So if the adversary is to output a triple F_{1,i_1}, F_{1,i_2}, F_{1,i_3} satisfying $F_{1,i_2} = F_{1,i_1} \cdot R$ and $F_{1,i_3} = F_{1,i_1} \cdot S$, then the only possibilities values for F_{1,i_1} are

$$(\{a_3 + a_7Y_1 \mid a_3, a_7 \in \mathbb{Z}_p\} \cap \{a_4X + a_8Y_2 \mid a_4, a_8 \in \mathbb{Z}_p\}) \setminus \{0\} = \varnothing \ .$$

Thus, the adversary cannot win when the oracle follows the rules of the formal game.

A.3 Real Game

Next, we argue that, with probability at least $1 - \frac{24(q+11)^2}{p}$, an adversary playing the formal game receives oracle query answers distributed identically to those it would have received in the real game.

We might imagine two ways the formal game could differ from the real game. The first would be for the oracle to give out a previously returned identifier when it should have selected a new one. This would happen if the adversary made a query on two rational functions and the result was formally identical to a previous rational function, but when evaluated on the specific values x, y_1, y_2, r, s, the two differed. Of course, this cannot happen. If two rational functions are identical, they will have the same value when evaluated.

The other way the formal game could differ from the real game would be for the oracle to give out a new identifier when it should have given an existing one. That is, if the oracle returned the identifier of a new rational function $F_1 = \frac{P_1}{Q_1}$ which was not formally equal to an existing one $F_2 = \frac{P_2}{Q_2}$ (that is, $P_1Q_2 \neq P_2Q_1$), but $F_1(x, y_1, y_2, r, s) = F_2(x, y_1, y_2, r, s)$.

This case is indeed possible, but we argue that it occurs with probability (over the selection of x, y_1, y_2, r, s) at most $\frac{24(q+11)^2}{p}$. Specifically, $F_1(x, y_1, y_2, r, s) = F_2(x, y_1, y_2, r, s)$ iff

$$P_1(x, y_1, y_2, r, s)Q_2(x, y_1, y_2, r, s) = P_2(x, y_1, y_2, r, s)Q_1(x, y_1, y_2, r, s) \ ,$$

[1] Note that the adversary is capable of incorporating ρ into the values $a_1, \ldots, a_{10}$, for example, setting $a_1 = 4\rho$ or $a_3 = \rho^{-5}$.

that is, iff

$$P_1(x, y_1, y_2, r, s)Q_2(x, y_1, y_2, r, s) - P_2(x, y_1, y_2, r, s)Q_1(x, y_1, y_2, r, s) = 0 \ .$$

So we see that the oracle only gives an incorrect reply to a query on $F_1 = \frac{P_1}{Q_1}$ and $F_2 = \frac{P_2}{Q_2}$ if x, y_1, y_2, r, s is a root of the polynomial $P_1Q_2 - P_2Q_1$. We bound the probability of x, y_1, y_2, r, s being a root based on the degree of the polynomial.

Specifically, for any $\frac{P_1}{Q_1}$ and $\frac{P_2}{Q_2}$ in L_1,

$$\begin{aligned}
\deg(P_1Q_2 - P_2Q_1) &\leq \max(\deg(P_1Q_2), \deg(P_2Q_1)) \\
&= \max(\deg(P_1) + \deg(Q_2), \deg(P_2) + \deg(Q_1)) \\
&\leq \max(4 + 2, 4 + 2) \\
&= 6 \ ,
\end{aligned}$$

so $P_1Q_2 - P_2Q_1$ will have at most 6 roots. The probability that a query for the group operation in $\mathbb{G}_1$ will return the wrong result is thus at most $\frac{6}{p}$. The case of queries for the group operation in $\mathbb{G}_2$ is similar and results in the same bound. If $\frac{P_1}{Q_1}$ and $\frac{P_2}{Q_2}$ are in list L_3 (i.e., they are the result of pairing queries), we obtain the following bounds.

$$\begin{aligned}
\deg(P_1Q_2 - P_2Q_1) &\leq \max(8 + 4, 8 + 4) \\
&= 12 \ ,
\end{aligned}$$

So the probability of that type of query being answered incorrectly is at most $\frac{12}{p}$.

Now since the adversary is initially given 11 identifiers and makes at most q queries, the number of distinct queries it can make for either the group operation in $\mathbb{G}_1$, the group operation in $\mathbb{G}_2$, or the pairing is at most $(q + 11)^2$. So the total probability of at least one query being answered incorrectly is at most

$$(q + 11)^2\frac{6}{p} + (q + 11)^2\frac{6}{p} + (q + 11)^2\frac{12}{p} = \frac{24(q + 11)^2}{p} \ .$$

Appendix B

Proofs for the Signature of Reputation Scheme

B.1 Unforgeability of the Signature Scheme

We distinguish the following four types of forgeries which the adversary may attempt.

- **Type 1 forgery.** In the forgery, the adversary uses a ρ^* value that never appeared. This will break the BB-HSDH assumption by a trivial reduction.
- **Type 2 forgery.** In the forgery, the adversary uses $\rho^* = \rho_j$, but there exists k, such that $r_k^* \neq r_{j,k}$.
  - Case 1: $r_k^* = \gamma$.

    This breaks the BB-CDH assumption.

    Suppose the simulator obtains a BB-CDH instance (see Section 2.2).

    The adversary commits to q messages to be signed. The simulator chooses the parameters of the signature scheme, such that the variables $\gamma, g, \widehat{g}$ inherit the corresponding variables from the BB-CDH instance. For all $1 \leq i \leq \ell$, the simulator also chooses $\widehat{u}_i = \widehat{u}^{\tau_i}$, such that it knows their discrete logs τ_i (base $\widehat{u}$). The remaining parameters are picked directly.

    In the q signatures returned to the adversary, the simulator uses $\rho = \rho_1, \ldots, \rho_q$ respectively. Although the simulator does not know the secret signing key γ, it clearly has enough information to compute the signatures shown to the signature adversary.

    When the adversary outputs a Type 2-Case 1 forgery, it contains the term $\widehat{u}_k^{\gamma}$. As the simulator knows the discrete log of $\widehat{u}_k$ base $\widehat{u}$, it can compute $\widehat{u}^{\gamma}$, thereby breaking the BB-CDH assumption.

– Case 2: $r_k^* \in \{r_{j,1}, \ldots, r_{j,\ell}, s_{j,1}, \ldots, s_{j,\ell}\} \backslash \{r_{j,k}\}$.

Similar to Case 1, this also breaks the BB-CDH assumption. In particular, by renaming variables and letting $q = 1$, the BB-CDH assumption immediately implies the following assumption henceforth referred to as BB-CDH-1. Given the tuple

$$g, \widehat{g}, g^r, \widehat{g}^r, \widehat{u}, \rho, \widehat{g}^{\frac{1}{r+\rho}} \quad ,$$

it is computationally infeasible to output $\widehat{u}^r$.

We now show that if an adversary can succeed in a Type 2-Case 2 forgery, we can build a simulator that breaks the above BB-CDH-1 assumption.

The adversary first commits to q messages to be signed. The simulator guesses the j and $k' \neq k$ such that $\rho^* = \rho_j$ and $r_k^* = r_{j,k'}$. The case when $r_k^* = s_{j,k''}$ $(1 \leq k'' \leq \ell)$ is similar, so here, without loss of generality, we prove the case for $r_k^* = r_{j,k'}$.

When choosing parameters, the simulator inherits the $g, \widehat{g}$ values from the BB-CDH-1 instance. The simulator lets $\widehat{u}_k = \widehat{u}$. For $i \neq k$, the simulator picks $\widehat{u}_i = \widehat{g}^{r_i}$. The simulator picks $f_1 = x_{j,k'}^{-1} g^\tau$ where $\tau \xleftarrow{R} \mathbb{Z}_p$. The simulator picks the remaining parameters directly.

Now the simulator computes the signatures for the q messages specified by the adversary at the beginning of the game. For all except the j-th (message, signature) pair, the simulator computes all other signatures directly.

For the j-th signature, the simulator builds the ρ value from the BB-CDH-Derived instance into the signature: $\rho_j = \rho$. In addition, it builds the r value into the k'-th coordinate, that is, the simulator implicitly lets $r_{j,k'} = r$. Although the simulator does not know r, it can compute the term $\widehat{u}_{k'}^r$ as it knows the discrete log of $\widehat{u}_{k'}$ base $\widehat{g}$. In addition, the term $(x_{j,k'} f_1)^r = (g^r)^\tau$ can also be computed partly due to the way f_1 was chosen earlier. It is clear that the rest of the signature can be computed directly.

If the adversary outputs a forgery of this type, the forged signature contains $\widehat{u}_k^r = \widehat{u}^r$, thereby breaking the BB-CDH-1 assumption.

– Case 3: $r_k^* \notin \{r_{j,1}, \ldots, r_{j,\ell}, s_{j,1}, \ldots, s_{j,\ell}\}$, and and $r^* \neq \gamma$. This breaks the BB-HSDH assumption. In particular, by renaming variables, and letting $q = 2\ell + 1$, the BB-HSDH assumption states that given

$$\widehat{g}, \widehat{g}^\rho, \widehat{u}, g, g^\rho, \gamma, g^{\frac{1}{\rho+\gamma}}, (r_i, \widehat{g}^{\frac{1}{\rho+r_i}})_{1 \leq i \leq \ell}, (s_i, \widehat{g}^{\frac{1}{\rho+s_i}})_{1 \leq i \leq \ell}$$

it is hard to output $(g^{r^*}, \widehat{u}^{r^*}, g^{\frac{1}{\rho+r^*}})$, where $r^* \notin \{r_1, \ldots, r_\ell, s_1, \ldots, s_\ell, \gamma\}$. The simulator obtains this instance, and performs the following interactions with the adversary. The adversary first commits to q messages to be signed. Now the

simulator picks the parameters of the signature scheme to inherit the $g, \widehat{g}, \gamma$ variables from the above BB-HSDH instance. It picks $h = g^{\tau_1}, \widehat{g}_0 = \widehat{g}^{\tau_2}$, so that the simulator knows the exponents τ_1, τ_2, and can compute h^ρ and $\widehat{g}_0^\rho$ from g^ρ and $\widehat{g}^\rho$ respectively. For all $1 \leq i \leq \ell$, the simulator picks $\widehat{u}_i = \widehat{u}^{\mu_i}$. The simulator picks the remaining parameters directly.

The simulator guesses the j in which $\rho^* = \rho_j$. In the j-th signature returned to the adversary, the simulator uses the ρ and $\{r_i, s_i\}_{1 \leq i \leq \ell}$ values from the above BB-HSDH instance. It is not hard to see that the simulator has sufficient information to compute a signature for the j-th message. For all other $q-1$ (message, signature) pairs, the simulator computes their signatures directly.

If the adversary can succeed in a forgery of this case, the simulator can obtain the tuple $(g^{r_k^*}, \widehat{u}_i^{r_k^*} = (\widehat{u}^{r_k^*})^{\mu_k}, \widehat{g}^{\frac{1}{\rho + r_k^*}})$, thereby solving the above BB-HSDH instance.

- **Type 3 forgery**. In the forgery, the adversary uses $\rho^* = \rho_j$, $r_j^* = r_{j,i}$ for all $1 \leq i \leq \ell$, but there exists k, such that $s_k^* \neq s_{j,k}$. The proof is similar to Type 2 forgery.

- **Type 4 forgery**. In the forgery, the adversary uses $\rho^* = \rho_j$, $r_j^* = r_{j,i}$ and $s_j^* = s_{j,i}$ for all $1 \leq i \leq \ell$, but there exists a k such that $x_k^* \neq x_{j,k}$. This breaks the SCDH assumption through the following reduction. The simulator is given an SCDH instance (see Section 2.2), and performs the following interactions with the adversary.

  The adversary first commits to q messages to be signed. The simulator guesses the j in which $\rho^* = \rho_j, \forall i : r_i^* = r_{j,i}$ and $s_i^* = s_{j,i}$. When setting up parameters of the signature scheme, the simulator inherits the $\widehat{g}, g, h$ value from the above SCDH instance. For all $1 \leq i \leq \ell$, it lets $\widehat{u}_i = \widehat{u}^{\mu_i}, \widehat{v}_i = \widehat{v}^{\nu_i}$. The simulator also lets $f_1 = x_{j,k}^{-1} g^\tau$, and $f_2 = x_{j,k}^{-1} h^\omega$where $\tau, \omega \xleftarrow{R} \mathbb{Z}_p$.

  Now the simulator constructs signatures on the q specified messages and return them to the adversary. Except for the j-th (message, signature) pair, the simulator constructs all other (message, signature) pairs directly.

  For the j-th signature, the simulator uses the following strategy. It builds the ρ value from the SCDH instance into the j-th signature, that is $\rho_j = \rho$. In addition, it builds the r, s values from the SCDH instance into the k-th coordinate, that is, $r_{j,k} = r, s_{j,k} = s$. Although the simulator does not know the values of r, s, it knows or can compute all of the following terms in the signature. In particular, $(x_{j,k} f_1)^r = (g^r)^\tau$, $(x_{j,k} f_2)^s = (h^s)^\omega$. $\widehat{u}_k^r$ and $\widehat{v}_k^s$ can be computed as the simulator knows their discrete logs base $\widehat{u}$ and $\widehat{v}$ respectively. All the remaining terms are trivially computable.

  If the adversary success in a Type 3 forgery, the resulting signature contains $(x^*, (x^* f_1)^r, (x^* f_2)^s)$ where $x^* \neq x_{j,k}$. The simulator can thereby compute z, z^r, z^s, where $z = x^* x_{j,k}^{-1} \neq 1$, by dividing $(x^*, (x^* f_1)^r, (x^* f_2)^s)$ and $(x_{j,k}, (x_{j,k} f_1)^r, (x_{j,k} f_2)^s)$ coordinate-wise. This clearly breaks the SCDH assumption.

B.2 IK-CPA Security: Definition and Proof

IK-CPA security was first defined by Bellare et. al. [11]. The IKEnc described in Section 2.2 has IK-CPA security, that is, no polynomial-time adversary has more than negligible advantage in the following game:

Setup. The challenger returns to the adversary the public parameters of the encryption system $\mathsf{params}_{\mathrm{ike}}$, and two user public keys $\mathsf{upk}_{\mathrm{ike},0}$ and $\mathsf{upk}_{\mathrm{ike},1}$.

Challenge. The adversary submits two messages msg_0 and msg_1. The challenger flips a random coin b, and returns $\text{IKEnc.Enc}(\mathsf{params}_{\mathrm{ike}}, \mathsf{upk}_{\mathrm{ike},b}, \mathsf{msg}_b)$ to the adversary.

Guess. The adversary outputs a guess b' of b. The adversary wins the game if $b' = b$.

Remark 1. *The above security definition should still hold when* $\mathsf{upk}_{ike,0} = \mathsf{upk}_{ike,1} = \mathsf{upk}_{ike}$. *In this case, the security definition is equivalent to the standard IND-CPA security (under a specific user public key* upk_{ike}*).*

Proof. Consider the following hybrid sequence. In Game 0, the challenger encrypts msg_0 under $\mathsf{upk}_{\mathrm{ike},0}$ in the challenge stage. In Game R, the challenger returns to the adversary a random ciphertext in the challenge stage, that is, $(R_1, R_2, R_3) \xleftarrow{R} \mathbb{G}^3$. In Game 1, the challenger encrypts msg_1 under $\mathsf{upk}_{\mathrm{ike},1}$ in the challenge stage.

Below we prove that Game 0 is computationally indistinguishable from Game M. (The indistinguishability between Game 1 and Game M is similar, and hence omitted.)

Suppose a simulator obtains the following DLinear instance:

$$f, h, A, B \xleftarrow{R} \mathbb{G}, \quad f^r, h^s, T$$

It tries to distinguish whether $T \xleftarrow{R} \mathbb{G}$ or $T = A^r B^s$. See Definition 8 for more details on this DLinear variant.

Now the simulator sets up the public parameters of the encryption scheme to be f, h. It chooses $\mathsf{upk}_{\mathrm{ike},0} = (A, B)$, and it picks $(\mathsf{upk}_{\mathrm{ike},1}, \mathsf{usk}_{\mathrm{ike},1})$ by directly calling the IKEnc.GenKey algorithm. In the challenge stage, the adversary submits two messages msg_0 and msg_1. The simulator returns the following ciphertext to the adversary:

$$\mathsf{msg}_0 \cdot T, f^r, h^s$$

It is not hard to see that if $T = A^r B^s$, then the above simulation is identical to Game 0. Otherwise, it is identical to Game R. □

B.3 Definitions and Proofs for the Unblinded Scheme

B.3.1 Definitions

Voter anonymity. No polynomial-time adversary has more than negligible advantage in the following game.

Setup. The challenger gives the adversary all users' rcvkey and vpk. At this stage, the challenger retains all vsk to itself.

Corrupt. The adversary adaptively corrupts a user by learning its secret voting key vsk.

Vote. The adversary requests an unblinded vote from an uncorrupted voter to a recipient.

Challenge. The adversary submits two uncorrupted voters j_0^* and j_1^*, and a recipient i. The adversary must not have previously queried a vote from either j_0^* or j_1^* to i. The challenger flips a random coin b, and returns an unblinded vote from j_b^* to i.

ShowRep. The adversary specifies a signer i, and a list of voters $j_1, \ldots, j_c$, and signer i constructs rep based on votes from these voters. Notice that this may involve votes from j_0^* or j_1^* to i. rep is returned to the adversary.

Guess. The adversary outputs a guess b' of b, and wins the game if $b' = b$.

Vote unforgeability. No polynomial-time adversary has more than negligible advantage in the following game.

Setup. The challenger gives the adversary all users' rcvkey and vpk.

Corrupt. The adversary adaptively corrupts a user by learning its vsk.

Vote. The adversary requests a vote from an uncorrupted voter to a recipient.

ShowRep. The adversary specifies a signer i, and a list of voters $j_1, \ldots, j_c$. User i constructs rep based on votes from these voters, and returns it to the adversary.

Forge. The adversary outputs a vote from an uncorrupted user j^* to a recipient i^*. The adversary wins if the vote is correct, and it has not previously queried a vote from j^* to i^*.

Reputation anonymity. No polynomial-time adversary has more than negligible advantage in the following game.

Setup. The challenger generates n users, and reveal all users' keys including rcvkey, votekey to the adversary.

Challenge. The adversary chooses a user i^*, and a list of c voters $j_1, \ldots, j_c$. The challenger flips a random coin b, and depending on the value of b, it returns to the adversary either faithfully constructed rep, or a list of random numbers.

Guess. The adversary outputs a guess b' of b, and wins the game if $b' = b$.

B.3.2 Reputation Anonymity Proof

We prove the DLinear based instantiation.

Definition 8 (n-DLinear)**.** *Given* $(g, h, z_{1,1}, z_{1,2}, z_{2,1}, z_{2,2}, \ldots, z_{n,1}, z_{n,2}) \xleftarrow{R} \mathbb{G}^{2n+2}$*, and* g^r, h^s*, where* $r, s \xleftarrow{R} \mathbb{Z}_p$*, it is computationally infeasible to distinguish the following tuple from a completely random tuple:*

$$T = (z_{1,1}^r z_{1,2}^s,\ z_{2,1}^r z_{2,2}^s,\ \ldots,\ z_{n,1}^r z_{n,2}^s)$$

Proof. We now prove that the n-DLinear assumption is implied by the DLinear assumption. Let $0 \leq d \leq n$, let $\Gamma_i = z_{i,1}^r z_{i,2}^s$. Define a hybrid sequence: in the Game d ($0 \leq d \leq n$), the challenger gives the adversary $g, h, g^r, h^s, z_{1,1}, z_{1,2}, z_{2,1}, z_{2,2}, \ldots, z_{n,1}, z_{n,2},$ and the following tuple:

$$*, *, \ldots, *, \Gamma_{d+1}, \ldots, \Gamma_n$$

where each $*$ denotes an independent random element from $\mathbb{G}$.

Due to the hybrid argument, it suffices to show that no PPT adversary can distinguish any two adjacent games.

We now show that no PPT adversary can distinguish between Game d and Game $d-1$, where $1 \leq d \leq n$. Suppose a simulator gets a DLinear instance g, h, f, g^r, h^s, X, it tries to tell whether $X = f^{r+s}$ or $X \xleftarrow{R} \mathbb{G}$. It picks $z_{d,1} = f, z_{d,2} = fh^\tau$. For all $i \neq d$, the simulator picks $z_{i,1} = g^{\omega_i}, z_{i,2} = h^{\mu_i}$. Now the simulator gives the adversary $g, h, g^r, h^s, z_{1,1}, z_{1,2}, z_{2,1}, z_{2,2}, \ldots, z_{n,1}, z_{n,2},$ and the following tuple:

$$*, *, \ldots, *, X \cdot (h^s)^\tau,\quad (g^r)^{\omega_{d+1}}(h^s)^{\mu_{d+1}}, \ldots, (g^r)^{\omega_n}(h^s)^{\mu_n}$$

Clearly, the above game is equivalent to Game $d-1$ if $X = f^{r+s}$. Otherwise, it is equivalent to Game d. □

Now we build a simulator that leverages an adversary against reputation anonymity to break the n-DLinear assumption. When choosing all users' upk and usk values, the simulator inherits the $z_{i,k}$ values from the n-DLinear assumption, where $1 \leq i \leq n, k \in \{1, 2\}$. It picks $x_{i,1} = g^{\tau_{i,1}}, y_{i,1} = g^{\omega_{i,1}}$, and $x_{i,2} = h^{\tau_{i,2}}, y_{i,2} = h^{\omega_{i,2}}$ for all $1 \leq i \leq n$. It picks all other parameters in upk and usk directly.

In the challenge phase, the simulator computes rep as below:

$$\forall 1 \leq j \leq m : u_{j,1}^r u_{j,2}^s = (x_{i,1}^{\alpha_j} y_{i,1}^{\beta_j})^r (x_{i,2}^{\alpha_j} y_{i,2}^{\beta_j})^s \cdot T_j$$

where T_j is inherited from the n-DLinear instance. As the simulator knows the discrete-log of $x_{i,1}$ and $y_{i,1}$ base g, and the discrete-log of $x_{i,2}$ and $y_{i,2}$ base h, the simulator can compute the term $(x_{i,1}^{\alpha_j} y_{i,1}^{\beta_j})^r (x_{i,2}^{\alpha_j} y_{i,2}^{\beta_j})^s$.

It is not hard to see that if T is a true n-DLinear tuple, the above constructed rep is faithful. Otherwise, it is a random tuple.

B.3.3 Voter Anonymity Proof

Game 1: answering ShowRep queries at random. First, modify the voter anonymity game such that when the adversary makes ShowRep queries, the challenger simply returns a list of random numbers. Answering ShowRep queries randomly does not affect the adversary's advantage in winning the voter anonymity game.

To see why, observe that it is computationally infeasible to distinguish between Game 1 and the real voter anonymity game. This can concluded from reputation anonymity and a simple hybrid argument.

Reduction to the DLinear assumption. We can now reduce voter anonymity to the DLinear assumption.

Below, we prove the real-or-random version of voter anonymity.

Notice that DLinear implies that the following problem is hard: Given

$$g, h, g^{\alpha}, h^{\beta}, u_1, v_1, u_2, v_2, T$$

it is computationally infeasible to distinguish whether $T = (u_1^{\alpha} v_1^{\beta}, u_2^{\alpha} v_2^{\beta})$ or $T \xleftarrow{R} \mathbb{G}^2$. In fact, this is a special case of the n-DLinear assumption mentioned above, with $n = 2$.

- **Setup**. The simulator obtains the above 2-DLinear instance. It inherits the parameters g, h from the 2-DLinear instance. It guesses the challenge voter j^* and recipient i^*. It picks $x_{i^*,k} = u_k, y_{i^*,k} = v_k$ for $k \in \{1, 2\}$. For all $i \neq i^*$, pick $x_{i,k} = g^{\tau_{i,k}}, y_{i,k} = h^{\omega_{i,k}}$ for $k \in \{1, 2\}$. In addition, the simulator lets $\alpha_{j^*} = \alpha$, $\beta_{j^*} = \beta$. The remaining parameters are picked directly. It is not hard to see that the simulator can compute all users vpk and rcvkey, which the simulator releases to the adversary at the beginning of the game.

- **Corrupt**. If the targeted user is j^*, abort. Otherwise, return to the adversary the user's vsk.

- **Vote**. If the vote queried is from j^* to i^*, abort. Otherwise, if the voter is j^* and the recipient is $i \neq i^*$, compute the vote as follows:

  $$[(g^{\alpha})^{\tau_{i,k}} (h^{\beta})^{\omega_{i,k}} z_{j^*,k}]_{k \in \{1,2\}}$$

  Else if the voter is not j^*, compute the vote directly.

- **ShowRep**. Return a random tuple.

- **Challenge**. If the challenger voter and recipient are not j^* and i^*, abort. Otherwise, return the following vote:

$$(T_1 z_{j^*,1}, T_2 z_{j^*,2})$$

Clearly, if $T = (T_1, T_2)$ is a true 2-DLinear instance, the above vote is correctly constructed. Otherwise, it is a random pair.

B.3.4 Vote Unforgeability Proof

- **Setup**. The simulator obtains a CDH instance g, g^α, h. It tries to output h^α.

The simulator guesses the voter j^* and recipient i^* in the forged vote output by the adversary. It implicitly lets $\alpha_{j^*} = \alpha$. It picks $x_{i^*,k} = h^{\tau_{i^*,k}}$ where $k \in \{1, 2\}$. For any user $i \neq i^*$, the simulator picks $x_{i,k} = g^{\tau_{i,k}}$ where $k \in \{1, 2\}$. The simulator picks the remaining parameters directly. It is not hard to see that the simulator can compute all users vpk and rcvkey, which the simulator releases to the adversary at the beginning of the game.

- **Corrupt**. If the targeted user is j^*, abort. Otherwise, give away the user's vsk to the adversary.

- **Vote**. If the requested vote is from j^* to i^*, abort. If the requested vote is from j^* to $i \neq i^*$, the simulator computes the vote as follows:

$$[(g^\alpha)^{\tau_{i,k}} y_{i,k}^{\beta_{j^*}} z_{j^*}]_{k \in \{1,2\}}$$

If the requested vote is from $j \neq j^*$ to any user i, the simulator computes the vote directly.

- **ShowRep**. Return a random tuple. Like in the voter anonymity game, this change should not affect the adversary's probability in winning the vote unforgeability game.

- **Forge**. When the adversary outputs a vote from j^* to i^*, the simulator can compute h^α as below. First, denote the vote as (u_1, u_2). Now, compute h^α as below:

$$(u_1 z_{j^*,1}^{-1} y_{i^*,1}^{-\beta_{j^*}})^{\frac{1}{\tau_{i^*,1}}}$$

B.4 Proofs for the Full Scheme

Theorem B.4.1. *The algorithms* Setup, GenCred, GenNym, Vote, SignRep, *and* VerifyRep *defined in Section 2.3 constitute a* correct, receiver anonymous, voter anonymous, signer anonymous, *and* sound *scheme for signatures of reputation.*

In this section, we provide proofs for each of the four security properties defined in Section 2.1. We begin by proving a series of lemmas we will need.

In the three privacy games (receiver, voter, and signer anonymity), the adversary outputs a challenge, which varies from game to game, and challenger responds based on a coin flip b. In the proofs in this section, it will be convenient to refer to each of these as an oracle query the adversary can make, so we define the following additional oracle queries which correspond to the challenge stage of each game:

Ch_RecvAnon. On input (i_0^*, i_1^*), select $b \xleftarrow{R} \{0,1\}$ and respond with $\mathsf{nym}^* \leftarrow \textsc{GenNym}(\mathsf{params}, \mathsf{cred}_{i_b^*})$.

Ch_SignerAnon. On input $(j_0^*, j_1^*, \mathsf{nym}^*)$, select $b \xleftarrow{R} \{0,1\}$ and respond with $\mathsf{vt}^* \leftarrow \textsc{Vote}(\mathsf{params}, \mathsf{cred}_{j_b^*}, \mathsf{nym}^*)$.

Ch_VoterAnon. On input $(i_0^*, i_1^*, V_0^*, V_1^*, \mathsf{msg})$, select $b \xleftarrow{R} \{0,1\}$ and respond with $\Sigma_b^* \leftarrow \textsc{SignRep}(\mathsf{params}, \mathsf{cred}_{i_b^*}, V_b^*, \mathsf{msg})$.

We now go to prove the first lemma we will need.

B.4.1 Traceability

Intuitively, traceability means that all nyms, votes and signatures of reputation must be traceable to registered recipients and voters.

In the following, we refer to an additional opening algorithm OpenSigRep which works exactly like OpenNym and OpenVote. That is, it uses the extractor key xk to obtain the rcvkey of the signer from the commitment in the NIZK within the signature.

Lemma B.4.2. *No polynomial-time adversary has more than negligible advantage in the following game.*

Setup: The challenger runs the Setup algorithm, registers n users, and returns params and all users' credentials to the adversary.

Forge: The adversary wins the game if one of the following occurs:

- The adversary outputs a valid nym^* and $\textsc{OpenNym}(\mathsf{params}, \mathsf{openkey}, \mathsf{nym}^*) = \perp$.
- The adversary outputs a valid vote vt^* and $\textsc{OpenVote}(\mathsf{params}, \mathsf{openkey}, \mathsf{vt}^*) = \perp$.
- The adversary outputs a valid signature of reputation Σ^* and $\textsc{OpenSigRep}(\mathsf{params}, \mathsf{openkey}, \Sigma^*) = \perp$.

In the above, "valid" means that the nym^*, vt^* or Σ^* passes the corresponding verification algorithm.

We also refer to the above as nym traceability, vote traceability, signature of reputation traceability respectively.

Proof. We prove the case for signature of reputation traceability. The other cases are similar if not easier. There are 2 possible cases if $\text{OpenSigRep}(\mathsf{params}, \mathsf{openkey}, \Sigma^*) = \perp$:

- Case 1: The OpenSigRep algorithm uses the extractor key xk of the GS proof system to extract the signer's *pub_cred* $= (\mathsf{rcvkey}, \mathsf{vpk}, \mathsf{vk}_{\text{bb}}, \mathsf{upk}_{\text{ike}})$, and a certificate on the above tuple. The *pub_cred* extracted is not among the registered users.
- Case 2: Use xk to extract a list of votes, and now further use OpenVote on these votes to extract a set S of c distinct voters, more specifically, for each $j \in S$, extract $\textit{pub\_cred}_j = (\mathsf{rcvkey}_j, \mathsf{vpk}_j, \mathsf{vk}_{\text{bb}j}, \mathsf{upk}_{\text{ike}j})$ and a certificate for each $\textit{pub\_cred}_j$. There exists a $j \in S$ such that $\textit{pub\_cred}_j$ is not among the registered users.

If either of the above cases is true, we can build a simulator that breaks the security of the certification scheme, or more specifically, the existential unforgeability under the weak chosen message attack (henceforth referred to as weak EF-CMA security).

At the beginning of the game, the simulator picks $\textit{pub\_cred}_i = (\mathsf{rcvkey}_i, \mathsf{vpk}_i, \mathsf{vk}_{\text{bb},i}, \mathsf{upk}_{\text{ike},i})$ for all $1 \leq i \leq n$, and the corresponding secret keys $\mathsf{vsk}_i, \mathsf{sk}_{\text{bb},i}, \mathsf{usk}_{\text{ike},i}$. The simulator submits all $\textit{pub\_cred}_i$ $(1 \leq i \leq n)$ to the weak EF-CMA challenger $\mathcal{C}$. The EF-CMA challenger now returns to the simulator the public verification key $\mathsf{vk}_{\text{cert}}$ of the certification scheme, as well as n certificates on the submitted messages. With this, the simulator has chosen the $\mathsf{vk}_{\text{cert}}$ for our reputation system, as well as the user credentials for n users. The simulator picks the other required system parameters directly.

The simulator now releases all users' secret credentials to a traceability adversary, which outputs a forgery consisting of a signature of reputation Σ^*. No matter which of the above case is true, the simulator is able to extract a *pub_cred* that does not match any registered user, and a certificate for *pub_cred*. In this way, the simulator has forged a certificate on a new message, thereby breaking the weak EF-CMA security of the certification scheme. □

B.4.2 Non-frameability

Lemma B.4.3. *No polynomial-time adversary has more than negligible advantage in the following game.*

Setup: The challenger runs the Setup algorithm, registers n users, and returns params to the adversary.

Query: The adversary adaptively makes *Corrupt*, *Nym*, *Vote* and *SignRep* queries to the challenger. The adversary can also make any of the challenge queries, including *Ch_RecvAnon*, *Ch_SignerAnon*, *Ch_VoterAnon* queries.

Forge: The adversary wins the game if it succeeds in one of the following forgeries:

- *Nym forgery.* The adversary outputs a forged nym^*. nym^* has not been returned to the adversary by a previous *Nym* (or *Ch_RecvAnon*) query. In addition, $\textsc{OpenNym}(\mathsf{params}, \mathsf{openkey}, \mathsf{nym}^*) = i^*$, and i^* is not among those corrupted by the adversary through a *Corrupt* query.
- *Vote forgery.* The adversary outputs a forged vote vt^*. vt^* has not been returned to the adversary by a previous *Vote* (or *Ch_VoterAnon*) query, and vt^* opens to a voter j^* who has not been compromised by the adversary through a *Corrupt* query.
- *Signature of reputation forgery.* The adversary outputs a forged signature of reputation Σ^*. Σ^* has not been returned to the adversary by a previous *SignRep* (or *Ch_SignerAnon*) query, and Σ^* opens to a signer i^* who has not been compromised by the adversary through a *Corrupt* query.

Proof. We prove the case for nym non-frameability. The proofs for vote non-frameability and signature of reputation non-frameability are similar.

First, notice that the $(\mathsf{sk}_{\text{ots}}{}^*, \mathsf{vk}_{\text{ots}}{}^*)$ used in nym^* must agree one of those previously seen by the adversary in a *Nym*, *Vote* or *SignRep* query on the same user i^*. Otherwise, we can build a simulator that breaks the security of the BB-signature scheme, specifically, the existential unforgeability under the weak chosen message attack (henceforth referred to as weak EF-CMA security.) Groth used a similar argument in his group signature scheme [48]. Below we describe this reduction in detail.

The simulator guesses i^* at the beginning of the game. If the guess turns out to be wrong later in the game, the simulator aborts. The simulator guesses correctly with probability at least $1/n$, where n denotes the total number of registered users.

The simulator obtains a verification key $V = g^s \in \mathbb{G}$ from the BB signature challenger. The simulator picks user i^*'s verification key $\mathsf{vk}_{\text{bb},i^*}$ to be V. Notice that the simulator does not know the corresponding signing key $\mathsf{vk}_{\text{bb},i^*} = s$. The simulator picks the other elements of user i^*'s credential directly, and signs a certificate for it. The simulator need not know $\mathsf{vk}_{\text{bb},i^*} = s$ to produce the certificate, as the certificate signs $\mathsf{vk}_{\text{bb},i^*} = V$ rather than the secret signing key s.

The simulator chooses q random $(\mathsf{sk}_{\text{ots},1}, \mathsf{vk}_{\text{ots},1}), \ldots, (\mathsf{sk}_{\text{ots},q}, \mathsf{vk}_{\text{ots},q})$ pairs, and queries the BB signature challenger for signatures on $H(\mathsf{vk}_{\text{ots},1}), \ldots, H(\mathsf{vk}_{\text{ots},q})$. Whenever the adversary makes a *Nym*, *Vote*, or *SignRep* query, the simulator consumes one of these $(\mathsf{sk}_{\text{ots},i}, \mathsf{vk}_{\text{ots},i})$ where $1 \leq i \leq q$.

When the adversary outputs a forged nym^* with a $\mathsf{vk}^*_{\text{ots}}$ never seen before, the simulator uses, the extractor key to open the NIZK, and obtains a new pair $(H(\mathsf{vk}^*_{\text{ots}}), \text{BBSig.Sign}(H(\mathsf{vk}^*_{\text{ots}})))$. Due to the collision resistance of the hash function, $H(\mathsf{vk}^*_{\text{ots}}) \notin \{H(\mathsf{vk}_{\text{ots},1}), \ldots, H(\mathsf{vk}_{\text{ots},q})\}$ (except with negligible probability). This breaks the weak EF-CMA security of the BB signature scheme.

As the $\mathsf{vk}^*_{\text{ots}}$ used in nym^* agrees with one seen before (in a *Nym*, *Vote*, or *SignRep* query from i^*), nym^* must agree with a previously seen nym^* from user i^*. (nym^* cannot agree with

a previously seen vote or signature of reputation from i^*.) Otherwise, the nym^* would contain a one-time signature signed with $\mathsf{sk}^*_{\text{ots}}$ on a new message, where the message contains all of nym^* except the one-time signature part. This breaks the security of the one-time signature scheme through a simple reduction. □

B.4.3 Unforgeability

Lemma B.4.4. *No polynomial-time adversary has more than negligible advantage in the following game.*

Setup. The challenger sets up system parameters, registers n users, and returns params to the adversary.

Query. The adversary adaptively makes *Corrupt*, *Nym*, *Vote*, *SignRep* queries.

Forge. The adversary wins the game if it succeeds in either of the following types of forgeries:

- *Vote forgery.* The adversary outputs a vote vt^* such that OpenVote(params, $\mathsf{openkey}, \mathsf{vt}^*$) $= (j, i)$, where j has not been corrupted through a *Corrupt* query, and the adversary has not previously submitted a *Vote* query from user j to any nym that opens to i.
- *Signature of reputation forgery.* The adversary outputs a signature of reputation Σ^*. Suppose OpenSigRep opens Σ^* to the recipient i and c voters $j_1, \ldots, j_c$. There exists $j \in \{j_1, \ldots, j_c\}$ such that j has not been corrupted through a *Corrupt* query, and the adversary has not previously submitted a *Vote* query from user j to a nym which opens to i.

Proof. By reduction to vote unforgeability of the unblinded scheme. We will perform the simulation under a simulated crs. This means that we can no longer rely on the extractor key xk to open the NIZK. However, notice that we can also implement the open algorithms by decrypting the ciphertexts in the nyms, votes, and signatures of reputation. Notice that under a real crs, opening using xk or through decryption yield the same result due to the perfect soundness of NIZK.

Setup: The simulator chooses a simulated crs instead of a real crs, and it knows the simulation secret simkey. The simulator obtains all users' rcvkey and vpk from $\mathcal{C}$, the vote unforgeability challenger of the unblinded scheme. The simulator sets up the parameters of CCAEnc such that it knows the decryption key. The simulator picks all other system parameters directly. Notice that the simulator knows the $\mathsf{usk}_{\text{ike}}$ for all users.

Corrupt: The adversary specifies a user i to corrupt. The simulator forwards the query to $\mathcal{C}$, and obtains the user's vsk in return. The simulator returns the credential of user i to the adversary.

Nym, *SignRep*: It is not hard to see that the simulator can answer *Nym* and *SignRep* queries normally.

Vote: As the simulator has all users' $\mathsf{usk}_{\text{ike}}$, it is able to decrypt the ciphertext in the specified nym, and identify the recipient i. See Appendix B.4.4 for more details on how this step can be achieved.

Now the simulator forwards the voter j and the recipient i to $\mathcal{C}$, and obtains an unblinded vote from j to i. To compute the vote, the simulator encrypts the unblinded vote under IKEnc to obtain the term C_1. Then it computes C_2 normally. It uses the simulation secret simkey to compute the NIZK, and eventually, uses the one-time signature scheme to sign everything. It is not hard to see that a vote computed in this way is identically distributed as a real vote under a simulated crs.

Forge: Eventually, the adversary outputs a forgery. If the forgery is a vote vt^*, the simulator decrypts the IKEnc ciphertext in vt^* to obtain an unblinded vote U^*. Otherwise, if the forgery is a signature of reputation Σ^*, the simulator decrypts the CCAEnc ciphertext in Σ^* to obtain a list of unblinded votes, among which is U^*. If the adversary wins the vote unforgeability game, then U^* is from an uncorrupted voter j to a recipient i, and the adversary has never made a *Vote* query from j to a nym corresponding to i. This means that our simulator has broken the vote unforgeability of the unblinded scheme.

□

B.4.4 Alternative implementation of OpenNym

In all of the games, when the adversary makes a SignRep query (or a *Ch_SignerAnon* query), the challenger needs to check if the set of votes supplied by the adversary correspond to the recipient specified by the adversary. To do this, the challenger calls the OpenNym algorithm to trace the owners of the nyms that are included in the votes.

We now propose an alternative implementation of the OpenNym oracle.

First, we define a sub-routine called TestNym to test if a nym belongs to a specific user.

TestNym(params, nym, cred_i). The TestNym subroutine checks if a nym is owned by user i. Parse nym as $\mathsf{nym} = (\mathsf{vk}_{\text{ots}}, C, \Pi, \sigma_{\text{ots}})$, where $C = (C_0, C_1, C_1')$ consists of an encryption of user i's receiver key rcvkey_i (denoted C_0) and two random encryptions of $1 \in \mathbb{G}$ (denoted C_1 and C_1'). Let $\mathsf{usk}_{\text{ike},i}$ denote the secret decryption key of user i.

The TestNym algorithm first tests if the following equations are true:

$$\begin{aligned}
\text{IKEnc.Dec}(\mathsf{params}_{\text{ike}}, \mathsf{usk}_{\text{ike},i}, C) &= \mathsf{rcvkey}_i \\
\text{IKEnc.Dec}(\mathsf{params}_{\text{ike}}, \mathsf{usk}_{\text{ike},i}, C_1) &= 1 \\
\text{IKEnc.Dec}(\mathsf{params}_{\text{ike}}, \mathsf{usk}_{\text{ike},i}, C_1') &= 1
\end{aligned}$$

Next, The TESTNYM algorithm checks that the two encryptions of 1 are uncorrelated in the following sense. Let $C_1 = (c_1, c_2, c_3)$, and let $C'_1 = (c'_1, c'_2, c'_3)$. The algorithm makes sure that

$$e(c_2, c'_3) \neq e(c'_2, c_3) \quad . \tag{B.1}$$

If both of the above checks pass, the algorithm concludes that the nym's owner is user i.

Lemma B.4.5. *If* nym *is generated by user i, then* TESTNYM*(*params*,* nym*,* cred$_i$*) returns* 1 *(except negligible probability). In addition, there exists at most one $i \in \{1..n\}$ such that* TESTNYM(nym, cred$_i$) $= 1$.

Proof. The first direction is obvious: if nym is generated by user i, clearly, TESTNYM(params, nym, cred$_i$) returns 1 (except negligible probability).

For the other direction, assume for the sake of contradiction that there exist 2 users $i \neq j \in \{1..n\}$ such that TESTNYM(nym, cred$_i$) $= 1$, and TESTNYM(nym, cred$_j$) $= 1$. This means that there exist $\mathsf{upk}_{\text{ike},i} = (A_i, B_i) \neq \mathsf{upk}_{\text{ike},j} = (A_j, B_j)$, and $r_1, s_1, r_2, s_2 \in \mathbb{Z}_p$, and

$$E_{\mathsf{upk}_{\text{ike}},i}(1, r_1, s_1) = E_{\mathsf{upk}_{\text{ike}},j}(1, r_2, s_2)$$
$$E_{\mathsf{upk}_{\text{ike}},i}(1, r'_1, s'_1) = E_{\mathsf{upk}_{\text{ike}},j}(1, r'_2, s'_2)$$

Clearly, for the latter two terms in the ciphertext to be equal, $r_1 = r_2 = r$, and $s_1 = s_2 = s$. Similarly, $r'_1 = r'_2 = r'$, and $s'_1 = s'_2 = s'$. For the first term in the ciphertext to be equal, we obtain through basic algebra:

$$A_i^r B_i^s = A_j^r B_j^s \quad \Rightarrow \quad (A_i/A_j)^r = (B_j/B_i)^s$$

Similarly,

$$(A_i/A_j)^{r'} = (B_j/B_i)^{s'}$$

But this would break the second check in the TESTNYM algorithm (see Equation (B.1)). □

We now describe an alternative implementation of the OPENNYM algorithm which the simulator will use in the simulations. Let S_c denote the set of users that have been corrupted by the adversary thus far. Let $L = \{\mathsf{nym}_k, id_k\}_{1 \leq k \leq q}$ denote the list of nyms that have been returned to the adversary through a *Nym* query (or a *Ch_RecvAnon* query). nym_k is the nym returned to the adversary, and id_k denotes the id of the user specified in the query. (In the case of a *Ch_RecvAnon*, id_k should be the one of the two users specified in the query, depending on the challenger's coin.)

OPENNYM :

Step 1: for $i \in S_c$: if TESTNYM(nym, cred$_i$) $= 1$ then return i;

Step 2: if nym $\in L$: return the corresponding id

Step 3: return $\perp$

Remark 2. *This means that the adversary essentially has enough information to perform* OpenNym *itself, on any valid nym it is able to construct.*

Lemma B.4.6. *This alternative* OpenNym *procedure is correct in the real crs world, except with negligible probability.*

Proof. Due to nym traceability and unforgeability, except with negligible probability, either 1) the nym agrees with one previously seen by the adversary; or 2) the nym opens to a user within the adversary's coalition (where the open operation is performed using the extractor key.) Due to the perfect soundness of NIZK, for the second case, using the extractor key or TestNym to open the nym would produce the same result. □

Notice that the above alternative OpenNym implementation works in all games, even though each game may have a different type of challenge query.

B.4.5 Signer Anonymity

We perform the simulation in the simulated crs world. This shouldn't affect the adversary's advantage by more than negligible amount. Under a simulated crs, the NIZK has perfect zero-knowledge. Therefore, the only terms in the signature of reputation that can possibly reveal the signer is the encryption of the unblinded vote $\text{CCAEnc.Enc}(U_1, \ldots, U_c)$, and $\mathsf{rep} = \text{ShowRep}(U_1, \ldots, U_c)$.

In the challenge stage, the adversary submits two signers i_0 and i_1, and a list of votes for each signer. Let $\vec{U}_0 = (U_{0,1}, \ldots, U_{0,c})$, $\vec{U}_1 = (U_{1,1}, \ldots, U_{1,c})$ denote the set of distinct unblinded votes for i_0 and i_1 respectively. These unblinded votes may be obtained by decrypting the ciphertexts in the votes.

Now consider the following hybrid sequence:

Game 0: In Game 0, the challenger chooses user i_0 to answer the challenge query. That is,

$$\Sigma^* = \Big(\ldots \quad E_0 = \text{CCAEnc.Enc}(\vec{U}_0), \quad \mathsf{rep}_0 = \text{ShowRep}(\vec{U}_0), \quad \ldots \Big)$$

Game M: In Game M, the challenger encrypts the unblinded votes from i_0, but uses the unblinded votes from i_1 in the ShowRep algorithm. In addition, it uses the simulation secret simkey to construct the NIZK.

$$\Sigma^* = \Big(\ldots \quad E_0 = \text{CCAEnc.Enc}(\vec{U}_0), \quad \mathsf{rep}_1 = \text{ShowRep}(\vec{U}_1), \quad \ldots \Big)$$

Game 1: In Game 1, the challenger chooses user i_1 to answer the challenge query. That is,

$$\Sigma^* = \Big(\ldots \quad E_1 = \text{CCAEnc.Enc}(\vec{U}_1), \quad \mathsf{rep}_1 = \text{ShowRep}(\vec{U}_1), \quad \ldots \Big)$$

Game M and Game 1 are indistinguishable. We can prove this through a reduction to the security of the encryption scheme. We only require CPA security in this case. We now show that given an adversary that can distinguish Game M and Game 1, we can build a simulator that breaks the CPA security of the encryption scheme. The simulator obtains the public key of the CCAEnc scheme from an encryption challenger $\mathcal{C}$. The simulator sets up the parameters of our reputation system, such that the CCAEnc used in our reputation system agrees with those received from $\mathcal{C}$. The simulator picks all other parameters directly (under a simulated crs), and proceeds with the signer anonymity game as prescribed, except for the *Ch_SignerAnon* query. In the *Ch_SignerAnon* query, the simulator decrypts the IKEnc ciphertext in the submitted votes, and obtains two sets of unblinded votes: $\vec{U}_0 = (U_{0,1}, \ldots, U_{0,c})$ and $\vec{U}_1 = (U_{1,1}, \ldots, U_{1,c})$. The simulator submits the two sets of unblinded votes to the encryption challenger, and obtains $E_b = \text{CCAEnc.Enc}(\vec{U}_b)$. The simulator builds E_b and rep_1 into the challenge signature of reputation Σ^*, and uses the simulation secret to simulate the NIZK proofs. If the adversary can distinguish whether it is in Game M or Game 1, then the simulator would succeed in distinguishing which set of unblinded votes $\mathcal{C}$ encrypted. Notice that in the above, we modified the standard IND-CPA security game such that the simulator submits two sets of plaintexts (as opposed to two plaintexts) to the challenger $\mathcal{C}$. This can be derived from the standard IND-CPA security through a simple hybrid argument.

Game 0 and Game M are indistinguishable. By reduction to the reputation anonymity of the unblinded scheme.

The simulator obtains all users' $(\mathsf{rcvkey}, \mathsf{votekey})$ from $\mathcal{C}$, the challenger of the unblinded scheme. The simulator builds these into the user credentials of the full reputation system. The simulator picks all other parameters needed directly (under a simulated crs), and proceeds to interact with the adversary prescribed, except in the *Ch_SignerAnon* query.

When answering the *Ch_SignerAnon* query, the simulator first checks if all the votes correspond to the same receiver specified by the adversary. This can be done through the OpenNym algorithm defined in Appendix B.4.4, as the simulator knows all users' secret credentials. The simulator now decrypts the IKEnc ciphertext in the votes to obtain two sets of unblinded votes, $\vec{U}_0 = (U_{0,1}, \ldots, U_{0,c})$ and $\vec{U}_1 = (U_{1,1}, \ldots, U_{1,c})$.

Had we used a real crs, these unblinded votes must be traceable to a registered voter and recipient (except with negligible probability), due to the traceability of votes and the perfect soundness of NIZK proofs. Under a simulated crs, the unblinded votes must be traceable to a registered voter and recipient as well, since otherwise, we may build a simulator that distinguishes a simulated crs and a real crs. As the simulator knows all users' rcvkey and $\mathsf{votekey}$, the simulator can identify the voters from these unblinded votes through a brute force enumeration method:

If $U = \text{VoteUnblinded}(\mathsf{rcvkey}_i, \mathsf{votekey}_j)$, then U is an unblinded vote from j to i

As a result, the simulator obtains two sets of voters $\vec{j}_0 = (j_{0,1}, \ldots, j_{0,c})$, and $\vec{j}_1 = (j_{1,1}, \ldots, j_{1,c})$. The simulator now submits the two lists of voters to $\mathcal{C}$, and in return, obtains $\mathsf{rep}_b = \textsc{ShowRep}(\vec{U}_b)$. It builds E_0 and rep_b into the challenge signature of reputation Σ^*, and uses the simulation secret simkey to simulate the NIZK proofs. Clearly, if $b = 0$, then the above game is identical to Game 0; otherwise, it is identical to Game M.

B.4.6 Voter Anonymity

In the voter anonymity game, the adversary submits two voters j_0^* and j_1^* and a nym^* in the challenge stage, and obtains a vote from one of these voters j_b^* on the specified nym^*. nym^* must open to an uncorrupted user i^*. There are two places in the simulation that can leak information about the challenger's coin b. The first place is obviously the challenge vote. The second place is more obscure: if the adversary submits the challenge vote vt^* (or some correlated version of it) in a *SignRep* query, it learns a signature of reputation that encodes information about j_b^*. We start by proving that the adversary is not able to learn anything from the *SignRep* queries. To this end, we define the following hybrid game:

Game I. We modify the original voter anonymity game in the following way. Whenever the adversary makes a *SignRep* query, the challenger decrypts the IKEnc ciphertext in the votes and obtains a list of unblinded votes. Let c denote the number of distinct unblinded votes. The challenger now picks a random recipient, and random c voters. It computes a signature of reputation corresponding to the above recipient and voters. We henceforth refer to a signature of reputation constructed in this way as a random signature of reputation. Notice that in Game I, the *SignRep* queries contain no information about which voter was chosen in the *Ch_VoterAnon* query.

Lemma B.4.7. *Answering SignRep queries with random signatures of reputation does not affect the adversary's advantage in the voter anonymity game by more than a negligible amount.*

Proof. By reduction to signer anonymity. Let q denote the total number of *SignRep* queries where vt^* is involved. Consider the following hybrid sequence. In the d-th game ($0 \leq d \leq q$), the challenger truthfully answers the first k *SignRep* queries where vt^* is involved. For the remaining *SignRep* queries, the simulator returns random signatures of reputation. Due to the hybrid argument, it suffices to show that the $(d-1)$-th and d-th games are computationally indistinguishable, where $1 \leq d \leq q$.

Setup: The simulator obtains params and all users' credentials from the signer anonymity challenger $\mathcal{C}$.

Corrupt, *Nym*, *Vote*: Compute a result to these queries normally.

Ch_VoteAnon: Flip a random coin b and compute a vote from j_b^* normally.

SignRep: For the first $d-1$ *SignRep* queries, answer faithfully. For the d-th query, let $S = (\mathsf{vt}_1, \ldots, \mathsf{vt}_k)$ denote the list of votes specified by the adversary. The simulator first checks if all votes submitted correspond to the same recipient specified by the adversary by calling the OpenNym procedure defined in Section B.4.4.

As the simulator knows all users credentials, it can decrypt the IKEnc ciphertext in the votes and obtain a list of unblinded votes. Let c denote the number of distinct unblinded votes.

Now the simulator picks a random recipient i' and c distinct voters $j'_1, \ldots, j'_c$. It constructs c votes from $j'_1, \ldots, j'_c$ to i' respectively. Denote this set of votes as S'.

The simulator now submits the message msg, and two sets of votes S and S' to the signer anonymity challenger $\mathcal{C}$. In return, the simulator obtains a signature of reputation Σ, which the simulator passes along in response to the adversary's query.

Notice that if $\mathcal{C}$ returned a Σ corresponding to S, then the above simulation is identical to Game d. Otherwise, it is identical to Game $d-1$. Therefore, the adversary's advantage should differ only by a negligible amount in these two adjacent games. □

Remark 3. *In the above, the challenger computes a random signature of reputation by selecting c random voters $j_1, \ldots, j_c$ and a random recipient i, computing c votes from these voters to i, and then directly calling the* SignRep *algorithm to construct the signature of reputation.*

Under a simulated crs, the challenger can use the following alternative strategy: It computes c unblinded votes $U_1, \ldots, U_c$ from $j_1, \ldots, j_c$ to i respectively. Then, it calls CCAEnc.Enc *to encrypt these unblinded votes, and calls* ShowRep$(U_1, \ldots, U_c)$ *to construct* rep. *Finally, the challenger uses* simkey *to simulate the NIZK, and calls the one-time signature scheme to sign everything.*

Under a simulated crs, the signature of reputation computed in the above two ways are identically distributed.

Game II. Notice that in Game I, the challenger decrypts the IKEnc ciphertext in the votes to uncover the unblinded votes. In Game II, the challenger picks the parameters of the system such that it knows the decryption key to the CCAEnc scheme. Instead of decrypting the IKEnc ciphertext, the challenger decrypts the CCAEnc ciphertext instead, and counts the number distinct voters c. Then it returns to the adversary a random signature of reputation consisting of exactly c votes.

Game II is identically distributed as Game I under a real crs, due to the perfect soundness of the NIZK proofs and the traceability of the votes.

Later, under the simulated crs, the simulator sticks to decrypting the CCAEnc ciphertext for opening the votes.

We now show that the challenge vote vt^* does not reveal sufficient information for the adversary to distinguish whether vt^* comes from j_0^* or j_1^*. To demonstrate this, we will perform simulations under a simulated crs.

GameSim. The challenger now plays the above-defined Game II with the adversary under a simulated crs. This does not affect the adversary's advantage by more than a negligible amount.

We now show that the adversary's advantage in GameSim is negligible. There are two ciphertexts in the challenge vote vt^*, the IKEnc ciphertext C_1, and the CCAEnc ciphertext C_2. These two ciphertexts are the only places that may leak information about which voter is chosen for the challenge query.

We define the following hybrid sequence.

Game 0: The challenger chooses j_0^* for the challenge query.

Game M: When answering the challenge query, the challenger uses $\mathsf{votekey}_{j_1^*}$ to compute C_1 (through a homomorphic transformation as prescribed by the Vote algorithm). However, it calls CCAEnc.Enc to encrypt $x_{j_0^*}$, and produces C_2. The challenger now uses the simulation secret simkey to simulate the NIZK. Eventually, the challenger uses the one-time signature scheme to sign everything and returns the result to the adversary.

Game 1: The challenger chooses j_1^* for the challenge query.

Game M is indistinguishable from Game 1. By reduction to the security of the selective-tag CCA encryption scheme.

Setup: Simulator learns the public key of the CCAEnc scheme from a challenger $\mathcal{C}$ of the encryption scheme. The simulator selects $\mathsf{sk}_{\text{ots}}{}^*, \mathsf{vk}_{\text{ots}}{}^*$, computes the selected tag $\mathsf{tag}^* = H(\mathsf{vk}_{\text{ots}}{}^*)$, and commits tag^* to the challenger $\mathcal{C}$. The simulator sets up all other parameters as normal, and registers n users.

Corrupt, *Nym*, *Vote*: As the simulator knows all users' secret credentials, it can compute answers to these queries normally.

Ch_VoterAnon: The simulator obtains two voters j_0^*, j_1^* and a nym from the adversary. The simulator forward $x_{j_0^*}$ and $x_{j_1^*}$ to the encryption challenger $\mathcal{C}$, and gets back $C^* = \text{CCAEnc.Enc}(\mathsf{pk}_{\text{cca}}, x_{j_b^*}, \mathsf{tag}^*)$. It builds the ciphertext C^* into the challenge vote. Now the simulator uses $\mathsf{votekey}_{j_1^*}$ to compute the IKEnc ciphertext C_1, and uses simkey to simulate the NIZK proofs. Finally, it calls the one-time signature scheme to sign everything, and returns the resulting vote to the adversary.

SignRep: The query includes a list of votes. Check if all votes correspond to the same recipient specified by the adversary by calling the OpenNym procedure as defined in Appendix B.4.4.

Now the simulator needs to count the number of distinct voters. If the vote is the same as the challenge vote, consider that vote to be from either of the challenge voters. This will not affect the total count of distinct voters, due to the requirements of the voter anonymity game.

If the vote is not equal to the challenge vote, the simulator calls the decryption oracle of the CCAEnc scheme. The tag (under which the decryption oracle is called) must be different from the selected tag tag^*. We show this below in Lemma B.4.8.

The decryption oracle returns a set of x_j values that identify the set of voters. The simulator counts the number of distinct voters c, and constructs a random signature of reputation consisting of c distinct voters.

Clearly, if $\mathcal{C}$ returned $\text{CCAEnc.Enc}(\mathsf{pk}_{\text{cca}}, x_{j_0^*}, \mathsf{tag}^*)$, the above simulation is identically distributed as Game M. Otherwise, it is identically distributed as Game 1.

Lemma B.4.8. *In the above simulation, whenever the simulator queries the decryption oracle of the* CCAEnc*, the tag of the encryption differs from* tag^* *except with negligible probability.*

Proof. Due to the security of the one-time signature scheme, the vote (which is not equal to vt^*) must be signed under a key $\mathsf{sk}_{\text{ots}}{}' = \mathsf{sk}_{\text{ots}}{}^*$. Let $\mathsf{vk}_{\text{ots}}{}'$ denote the corresponding verification key. Then the tag used in the CCAEnc scheme $\mathsf{tag}' = H(\mathsf{vk}_{\text{ots}}{}')$ must be different from tag^* due to the collision resistance of the hash function. A more detailed proof of this can be found in Groth's group signature paper [48]. □

Game 0 is indistinguishable from Game M. We now show that if there exists an adversary that can distinguish Game 0 from Game M, we can build a simulator that breaks either the IND-CPA of the IKEnc scheme, or the vote anonymity of the unblinded scheme.

Recall that in the security definition of voter anonymity, the adversary can win the voter anonymity game in two cases depending on whether the recipient is corrupted or not.

Below, we build a simulator which guesses ahead of the game whether the challenge query will correspond to an uncorrupted recipient or a corrupted recipient. Depending on the guess, the simulator will use different strategies for the simulation. If later the simulator's guess turns out to be wrong, the simulator simply aborts. The simulator has probability at least a half of guessing right.

We now describe the simulator's strategy for each of the two cases.

Case 1: Uncorrupted recipient.

We build a reduction to the IND-CPA security of the IKEnc scheme. The simulator guesses upfront which recipient will be submitted in the *Ch_VoterAnon* query. Denote this challenge recipient as i^*. The simulator will abort if the guess later turns out to be wrong.

Setup: From an encryption challenger $\mathcal{C}$, the simulator obtains the public parameters $\mathsf{params}_{\text{ike}}$, and a user public key ${\mathsf{upk}_{\text{ike}}}^*$ which it tries to attack. The simulator lets this ${\mathsf{upk}_{\text{ike}}}^*$ to be user i^*'s user public key. For all other users, the simulator picks their $\mathsf{upk}_{\text{ike}}$ and $\mathsf{usk}_{\text{ike}}$ by directly calling the IKEnc.GenKey algorithm.

The simulator chooses parameters of the CCAEnc scheme such that it knows the decryption key. The simulator generates the remaining parameters directly.

Nym, *Vote*: The simulator can answer these queries directly.

Corrupt: If the adversary corrupts the i^*-th user, abort. Otherwise, return the secret credential for the specified user to the adversary.

SignRep: The simulator first checks if all votes correspond to the same recipient specified by the adversary, by using the OpenNym procedure defined in Section B.4.4. Next, the simulator calls the decryption algorithm of the CCAEnc scheme, to decrypt the CCAEnc ciphertext in the submitted votes. The simulator counts the number of distinct unblinded votes, and builds a random signature of reputation consisting of the same number of voters.

Ch_VoterAnon: The simulator gets two voters j_0^* and j_1^*, and a nym. If the nym does not open to i^*, abort. (OpenNym is implemented using the procedure described in Section B.4.4). The simulator now computes the unblinded votes $U_{j_0^*,i}$ and $U_{j_1^*,i}$ and submits them to the encryption challenger $\mathcal{C}$. The simulator gets back a ciphertext $\text{IKEnc.Enc}(\mathsf{params}_{\text{ike}}, {\mathsf{upk}_{\text{ike}}}^*, U_{j_b^*,i})$, this will be the C_1 ciphertext in the resulting vote. The simulator now computes the CCAEnc ciphertext directly on $x_{j_0^*}$, and simulates the NIZK proofs. Eventually, the simulator calls the one-time signature scheme to sign everything, and returns the resulting vote to the adversary.

It is not hard to see that if $\mathcal{C}$ encrypted $U_{j_0^*,i}$, then the above game would be identically distributed as Game 0. Otherwise, it is identically distributed as Game 1.

Case 2: Corrupted recipient.

By reduction to voter anonymity of the unblinded scheme.

Setup. The simulator obtains all users' rcvkey and vpk from the vote anonymity challenger $\mathcal{C}$ of the unblinded scheme. The simulator now guesses the challenge voters j_0^* and j_1^* that the adversary will specify in the *Ch_VoterAnon* query, as well as the challenge recipient i^*. If the simulator's guesses later turn out to be wrong,

the simulation simply aborts. Now, through *Corrupt* queries to $\mathcal{C}$, the simulator corrupts all users' vsk except for j_0^* and j_1^*. The simulator makes *VoteUnblinded* queries to $\mathcal{C}$, and obtains an unblinded vote from j_0^* and j_1^* to all users except i^*. The simulator picks the parameters of the CCAEnc such that it knows its secret decryption key. The remaining system parameters are picked directly.

Corrupt. If the adversary queries j_0^* or j_1^*, abort the simulation. Otherwise, return the specified user's credential to the adversary.

Nym. Compute directly.

Vote. The adversary submits a voter j and a nym which opens to i. The simulator can decide i by calling the alternative OpenNym algorithm (see Section B.4.4). If $j \in \{j_0^*, j_1^*\}$ and $i = i^*$, abort the simulation. Otherwise, the simulator has queried $\mathcal{C}$, and obtained an unblinded vote U from j to i. The simulator computes an IKEnc encryption of the unblinded vote U, by directly encrypting it. The simulator computes the CCAEnc ciphertext on x_j directly. The simulator uses simkey to simulate the NIZK proofs.

Ch_VoterAnon. The simulator calls the alternative OpenNym algorithm (see Section B.4.4) to decide the recipient i. The simulator now forwards the two specified voters j_0^* and j_1^* and the recipient i to $\mathcal{C}$, to obtain an unblinded vote U_b^* corresponding to one of the voters j_b^*. The simulator now encrypts the U_b^* by directly calling the IKEnc.Enc algorithm. The simulator computes the CCAEnc ciphertext on $x_{j_0^*}$ directly, and uses simkey to simulate the NIZK.

SignRep. The simulator calls the alternative OpenNym algorithm (see Section B.4.4) to check that all specified votes open to the specified recipient i. Now the simulator calls the decryption algorithm of the CCAEnc and obtains a set of voters. The simulator counts the number of distinct voters c, and computes a random signature of reputation with c voters. As we mentioned in Remark 3, the simulator only needs to know c unblinded votes for a random recipient i, to compute a random signature of reputation containing c voters.

B.4.7 Receiver Anonymity

By reduction to the IK-CPA security of the IKEnc scheme.

We perform the simulation under a simulated crs. This does not affect the adversary's advantage by more than a negligible amount.

Setup. The simulator guesses which two users i_0^* and i_1^* the adversary will submit in the *Ch_RecvAnon* query. The simulator obtains two user public keys $\mathsf{upk}_{\mathrm{ike},0}^*$ and $\mathsf{upk}_{\mathrm{ike},1}^*$ from the IK-CPA challenger $\mathcal{C}$. It lets i_0^*'s user public key to be $\mathsf{upk}_{\mathrm{ike},0}^*$, and i_1^*'s user public key to be $\mathsf{upk}_{\mathrm{ike},1}^*$. The simulator calls IKEnc.GenKey to generate the ($\mathsf{upk}_{\mathrm{ike}}$,

$\mathsf{usk}_{\text{ike}}$) pairs for all other users. As a result, the simulator knows all users' $\mathsf{usk}_{\text{ike}}$ except for i_0^* and i_1^*.

The simulator picks the parameters of the CCAEnc.Enc encryption scheme, such that it knows the secret decryption key. The simulator picks the remaining system parameters directly.

Corrupt. If the query is on i_0^* or i_1^*, abort. Otherwise, return the user's credential to the adversary.

Nym, *Vote.* Compute directly.

SignRep. In case any of the nyms specified is equal the challenge nym, abort.

Otherwise, the simulator calls the OpenNym procedure defined in Section B.4.4 to determine the recipient and check that all of the nyms correspond to the same recipient i as specified by the adversary.

The simulator decrypts the CCAEnc ciphertext in each vote and obtains a set of voters $j_1, \ldots, j_c$. With knowledge of all users $\mathsf{votekey}$ and rcvkey, the simulator can compute the unblinded votes from $j_1, \ldots, j_c$ to i. Now it computes the CCAEnc ciphertext and the rep parts of the signature based on the unblinded votes. The simulator uses the simkey to simulate the NIZK proofs.

Ch_RecvAnon. The adversary specifies two users, i_0^* and i_1^*. If i_0^* and i_1^* disagree with the simulator's guesses, abort. The simulator specifies $(\mathsf{rcvkey}_{i_0^*}, 1, 1)$ and $(\mathsf{rcvkey}_{i_1^*}, 1, 1)$ to $\mathcal{C}$ and obtains a challenge ciphertext
$C = \text{IKEnc.Enc}(\mathsf{params}_{\text{ike}}, \mathsf{upk}_{\text{ike},i_b^*}, (\mathsf{rcvkey}_{i_b^*}, 1, 1))$. Recall that the two encryptions of 1 are needed to rerandomize the ciphertext later when a voter performs homomorphic transformation on the ciphertext. (Due to the hybrid argument, the IK-CPA game may be modified such that the encryption adversary submits longer messages consisting of multiple elements in $\mathbb{G}$ in the challenge phase.) The simulator builds the challenge ciphertext C into the nym, and simulates the NIZK proofs. Finally, it calls the one-time signature scheme to sign everything, and returns the resulting nym to the adversary.

Guess. The adversary outputs a guess b'. The simulator outputs the same guess.

It is not hard to see that if the adversary succeeds in guess b' with more than negligible advantage, the simulator would have more than negligible advantage in the IK-CPA game.

B.4.8 Reputation Soundness

The adversary plays the reputation soundness game, and outputs a forged signature of reputation Σ^* at the end of the game. Suppose Σ^* signs the message msg^* and the reputation count c^*.

If the adversary wins the game, it is a requirement that the adversary has made a *SignRep* query on the message msg^* and reputation count c^*. Therefore, Σ^* cannot be equal to any signature of reputation returned by a *SignRep* query. Due to the traceability and non-frameability of signature of reputation, Σ^* must open to a signer i^* within the adversary's coalition, that is, a signer that has been corrupted through a *Corrupt* query.

Now apply OpenSigRep to open Σ^* to a set of c^* distinct voters. This fails with negligible probability due to the traceability of the signature of reputation. As $c^* > \ell_1 + \ell_2$, there must exist an uncorrupted voter j who voted for i^*, and the adversary has not made a *Vote* query from j to any nym that opens to i^*. But this breaks the unforgeability of signature of reputation.

B.5 Proofs for the Space Efficient Scheme

Theorem B.5.1. *The algorithms* Setup, GenCred, GenNym, Vote, SignRep′, *and* VerifyRep′ *constitute an* ε-sound *scheme for signatures of reputation.*

Proof. We prove this in the random oracle model through a reduction from the soundness of the regular scheme, which was proven in the previous section. Assume we have some adversary $\mathcal{A}$ trying to break the ε-soundness game. $\mathcal{A}$ will only be able to compute hash values by querying the random oracle H. As $\mathcal{A}$ runs, H records the queries $\mathcal{A}$ makes in a table and returns responses selected uniformly at random (except for repeated queries, in which case the previous value is returned).

Eventually $\mathcal{A}$ outputs a message msg and a forged signature of reputation Σ. Let $c = \text{VerifyRep}'(\mathsf{params}, \mathsf{msg}, \Sigma)$. Assume that $c \neq \perp$, $(1-\varepsilon)c > \ell_1 + \ell_2$, and the hashes and challenge set computation in Σ verify correctly. We will bound the probability that the all the votes in the challenge set verify.

The challenger looks through the hash values included and is able to find all their preimages based on the queries recorded by H. From these the challenger reconstructs the full hash tree and all c original leaf values $\omega_1, \ldots, \omega_c$. For each $1 \leq i \leq c$, the challenger verifies θ_i and ζ_i and checks that $R_{\rho_i} < R_{\rho_{i+1}}$. Let $c' \leq c$ be the number of these votes which pass both checks.

We distinguish two cases:

1. $c' \geq (1-\varepsilon)c$
2. $c' < (1-\varepsilon)c$

Case 1 must occur with probability less than or equal to some negligible function $\nu(\lambda)$, otherwise the challenger could output the $c' > \ell_1 + \ell_2$ valid votes and then it would be an adversary which would break the regular reputation soundness property.

We now bound the probability of all the challenge votes verifying in Case 2. Since the challenge indices were selected by evaluating H on unique inputs, I is a uniformly random

set of ℓ votes. So the probability that these are all among the c' valid votes is

$$\frac{c'}{c} \cdot \frac{c'-1}{c-1} \cdot \frac{c'-2}{c-2} \cdots \frac{c'-(\ell-1)}{c-(\ell-1)} = \prod_{i=0}^{\ell-1} \frac{c'-i}{c-i} \quad .$$

Since $c' < (1-\varepsilon)c$, this probability strictly less than

$$\prod_{i=0}^{\ell-1} \frac{(1-\varepsilon)c-i}{c-i} \quad .$$

We now show that the above value is at most $e^{-\lambda}$.

Since ℓ was selected as $\lceil \frac{\lambda}{\varepsilon} \rceil$, we may reason as follows.

$$\begin{aligned}
\ell = \left\lceil \frac{\lambda}{\varepsilon} \right\rceil &\geq \left\lceil \frac{\lambda}{-\log(1-\varepsilon)} \right\rceil \quad \text{(using the identity } \log(1+x) \leq x \text{ for } x > -1) \\
\implies \ell &\geq \frac{\lambda}{-\log(1-\varepsilon)} \\
\implies -\lambda \geq \ell \log(1-\varepsilon) &= \sum_{i=0}^{\ell-1} \log(1-\varepsilon) = \sum_{i=0}^{\ell-1} \log \frac{(1-\varepsilon)c}{c} \\
&\geq \sum_{i=0}^{\ell-1} \log \frac{(1-\varepsilon)c-i}{c-i} = \log \prod_{i=0}^{\ell-1} \frac{(1-\varepsilon)c-i}{c-i} \\
&\implies \log \prod_{i=0}^{\ell-1} \frac{(1-\varepsilon)c-i}{c-i} \leq \log e^{-\lambda} \\
&\implies \prod_{i=0}^{\ell-1} \frac{(1-\varepsilon)c-i}{c-i} \leq e^{-\lambda}
\end{aligned}$$

So the probability of all the challenge votes verifying and Case 2 occurring is strictly less than $e^{-\lambda}$.

So overall, the probability of all the challenge votes verifying is less than $e^{-\lambda} + \nu(\lambda)$, and thus the probability that Σ verifies is less than $e^{-\lambda} + \nu(\lambda)$. Since $\nu(\lambda)$ is negligible in λ, $e^{-\lambda} + \nu(\lambda)$ is also negligible in λ. □

Appendix C

Proofs for the Private Stream Searching Scheme

Here we provide proofs for Lemma 3.3.2, Lemma 3.3.3, and Theorem 3.3.6, which appeared in Section 3.3.

C.1 Singularity Probability of Pseudo-random Matrices

Lemma 3.3.2. *Let $G : \mathcal{K}_G \times \mathbb{Z} \times \mathbb{Z} \to \{0,1\}$ be a $(\omega_t, \omega_q, \epsilon/8)$-secure pseudo-random function family. Let $g = G_k$, where $k \stackrel{R}{\leftarrow} \mathcal{K}_G$. Let $\ell_F = o(\log(1/\epsilon))$ such that an $\ell_F \times \ell_F$ random $(0,1)$-matrix is singular with probability at most $\epsilon/4$. Then the matrix*

$$A = \Big[\, g(i,j) \,\Big]_{\substack{i=1,\ldots,\ell_F \\ j=1,\ldots,\ell_F}}$$

is singular with probability at most $\epsilon/2$.

Proof. We know that an $\ell_F \times \ell_F$ random $(0,1)$-matrix is singular with probability at most $\epsilon/4$. However, in our scheme, A is not a random matrix, but a matrix constructed using the pseudo-random function g. Thus, we need the additional proof step to show that the matrix A we constructed using the pseudo-random function g also satisfies the non-singular property with overwhelming probability, otherwise, we could break the pseudo-random function. This proof step is as follows.

Now assume for contradiction that the matrix A is singular with probability greater than $\epsilon/2$. Then we show that we can construct an adversary $\mathcal{B}$ (relative to the pseudo-random function family G) with $\mathsf{Adv}_{\mathcal{B}} > \epsilon/8$ with polynomial number of queries and polynomial time, thus contradicting the original assumptions of G.

To do so, we play the following game. We flip a coin $\theta \in \{0,1\}$ with a half and half probability, the adversary $\mathcal{B}$ is given one of two worlds in which it can make a number of

queries to a given oracle. If $\theta = 1$, $\mathcal{B}$ is given world one, where $g = G_k$, $k \xleftarrow{R} \mathcal{K}_G$, and the oracle responds to a query (i, j) with $g(i, j)$. If $\theta = 0$, the adversary $\mathcal{B}$ is given world two, where the oracle responds to a query (i, j) by picking a random function R mapping (i, j) to $\{0, 1\}$, i.e., by flipping a coin $b \in \{0, 1\}$ with a half and half probability and returning b (using a table of previous queries to ensure consistency). After a series of queries, the adversary $\mathcal{B}$ guesses which world it is in. The adversary $\mathcal{B}$ makes its guess using the following strategy: First, the adversary $\mathcal{B}$ constructs a matrix A by querying the oracle for all (i, j) where $i \in \{1, \ldots, \ell_F\}$ and $j \in \{1, \ldots \ell_F\}$; then the adversary $\mathcal{B}$ checks if A is singular. If yes, it guesses that it is in world one. If not, it guesses that it is in world two.

Thus, we can compute the advantage of such an adversary $\mathcal{B}$.

$$\mathsf{Adv}_{\mathcal{B}} = \left|\mathrm{P}\left(\mathcal{B}^g = 1\right) - \mathrm{P}\left(\mathcal{B}^R = 1\right)\right| = \left|\frac{1}{2}\mathrm{P}\left(\text{A is singular}|\theta = 1\right) - \frac{1}{2}\mathrm{P}\left(\text{A is singular}|\theta = 0\right)\right| .$$

From the above assumptions, $\mathrm{P}\left(\text{A is singular}|\theta = 1\right) > \epsilon/2$, and $\mathrm{P}\left(\text{A is singular}|\theta = 0\right) < \epsilon/4$, thus $\mathsf{Adv}_{\mathcal{B}} > \epsilon/8$, contradicting the original assumptions of G. □

C.2 Bounding False Positives

Lemma 3.3.3. *Given $\ell_F > m + 8\ln(2/\epsilon)$, let $\ell_I = O(m\log(t/m))$ and assume the number of matching files is at most m out of a stream of t. Then the probability that the number of reconstructed matching indices β is greater than ℓ_F is at most $\epsilon/2$.*

Proof. The number of reconstructed matching indices β equals the number of truly matching files plus the number of false positives from the reconstruction using the Bloom filter. Thus, we need to bound this number of false positives to be at most $\ell_F - m$.

The false positive rate ρ of the Bloom filter storing m entries is as follows [21].

$$\rho = \left(\frac{1}{2}\right)^{\frac{\ell_I \log 2}{m}} \tag{C.1}$$

Thus, the expectation of the number of false positives is ρt. For simplicity, let's set $\rho t = (\ell_F - m)/2$, which corresponds to the number of false positives filling about half the extra space on average. This choice is somewhat arbitrary, but it suffices to allow the proof to go through. So now $\ell_I = m(\log 2)^{-2}\log(\frac{2t}{\ell_F - m})$. Since ℓ_F is set to be linear in m, with $\ell_I = O(m\log(t/m))$ the expected number of false positives can be bounded far from ℓ_F.

Moreover, we can model the number of false positives with a binomial random variable X with rate parameter ρ and approximate it with a Gaussian centered at the expected number of false positives. From Chernoff bounds, we can derive that $\mathrm{P}\left(X > \ell_F - m\right) < \exp(-(\ell_F - m)/8)$. Thus, with $\ell_F > m + 8\ln(2/\epsilon)$, we can show that this probability is bounded by $\epsilon/2$. Thus, we show that the above lemma holds. □

C.3 Semantic Security

Here we provide a proof of the semantic security of the proposed private searching system assuming the semantic security of the Paillier cryptosystem. The proof is simple; in fact it proceeds in the same way as the proof of semantic security in Ostrovsky and Skeith's scheme [72]. The same proof applies whether we are using encrypted queries of the original form proposed by Ostrovsky and Skeith or the hash table queries we propose as an extension in Section 3.4.

Theorem 3.3.6. *If the Paillier cryptosystem is semantically secure, then the proposed private searching scheme is semantically secure according to the definition in Section 1.4.*

Proof. We assume there is an adversary $\mathcal{A}$ that can play the game described in the definition with non-negligible advantage ε in order to show that we then have non-negligible advantage in breaking the security of the Paillier cryptosystem.

First we initiate a game with the Paillier challenger, receiving public key n. We choose plaintexts $m_0, m_1 \in \mathbb{Z}_N$ to be simply $m_0 = 0$ and $m_1 = 1$. We return them to the Paillier challenger who secretly flips a coin β_1 and sends us $E(m_{\beta_1})$.

Now we initiate a game with $\mathcal{A}$ and send it the modulus n, challenging it to break the semantic security of the private searching system. The adversary sends us two sets of keywords, K_0 and K_1. We flip a coin β_2 and construct the query Q_{β_2} by passing K_{β_2} to QUERY. Next we replace all the entries in Q_{β_2} which are encryptions of one with $E(m_{\beta_1})$, re-randomizing each time by multiplying by a new encryption of zero. Note that with probability one half, $\beta_1 = 0$ and Q_{β_2} is a query that searches for nothing. In this case β_2 has no influence on Q_{β_2} since Q_{β_2} consists solely of uniformly distributed encryptions of zero. Otherwise, Q_{β_2} searches for K_{β_2}.

Next we give Q_{β_2} to $\mathcal{A}$. After investigation, $\mathcal{A}$ returns its guess β_2'. If $\beta_2' = \beta_2$, we let the guess for our challenge be $\beta_1' = 1$ and return it to the Paillier challenger. Otherwise we let $\beta_1' = 0$ and send it to the Paillier challenger.

Since $\mathcal{A}$ is able to break the semantic security of the private searching system, if $\beta_1 = 1$ the probability that $\beta_2' = \beta_2$ is $\frac{1}{2} + \varepsilon$, where ε is a non-negligible function of the security parameter n. If $\beta_1 = 0$, then $\mathrm{P}(\beta_2' = \beta_2) = \frac{1}{2}$, since β_2 was chosen uniformly at random and it had no bearing on the choice of β_2'. Now we may compute our advantage in our game with the Paillier challenger as follows.

$$\begin{aligned}
\mathrm{P}(\beta_1' = \beta_1) &= \mathrm{P}(\beta_1' = 1 | \beta_1 = 1)\frac{1}{2} + \mathrm{P}(\beta_1' = 0 | \beta_1 = 0)\frac{1}{2} \\
&= \left(\frac{1}{2} + \varepsilon\right)\frac{1}{2} + \frac{1}{2}\cdot\frac{1}{2} \\
&= \frac{1}{2} + \frac{\varepsilon}{2}
\end{aligned}$$

Since ε is non-negligible, so is $\frac{\varepsilon}{2}$. □

Appendix D

Details of the Stylometric Features

In this appendix, we provide some supplementary information about the stylometric features discussed in Section 5.3.

D.1 List of Function Words

Listed below are the 293 function words referenced in Table 5.1. The words "a" through "I" are shown here; the remainder of the list appears on the following page.

a	amidst	away	by	either	from
able	among	bar	can	enough	front
aboard	amongst	barring	certain	every	given
about	amount	be	circa	everybody	good
above	an	because	close	everyone	great
absent	and	been	concerning	everything	had
according	another	before	consequently	except	half
accordingly	anti	behind	considering	excepting	have
across	any	being	could	excluding	he
after	anybody	below	couple	failing	heaps
against	anyone	beneath	dare	few	hence
ahead	anything	beside	deal	fewer	her
albeit	are	besides	despite	fifth	hers
all	around	better	down	first	herself
along	as	between	due	five	him
alongside	aside	beyond	during	following	himself
although	astraddle	bit	each	for	his
am	astride	both	eight	four	however
amid	at	but	eighth	fourth	I

if	myself	over	so	till	whereas
in	near	part	some	time	wherever
including	need	past	somebody	to	whether
inside	neither	pending	someone	tons	which
instead	nevertheless	per	something	top	whichever
into	next	pertaining	spite	toward	while
is	nine	place	such	towards	whilst
it	ninth	plenty	ten	two	who
its	no	plethora	tenth	under	whoever
itself	nobody	plus	than	underneath	whole
keeping	none	quantities	thanks	unless	whom
lack	nor	quantity	that	unlike	whomever
less	nothing	quarter	the	until	whose
like	notwithstanding	regarding	their	unto	will
little	number	remainder	theirs	up	with
loads	numbers	respecting	them	upon	within
lots	of	rest	themselves	us	without
majority	off	round	then	used	would
many	on	save	thence	various	yet
masses	once	saving	therefore	versus	you
may	one	second	these	via	your
me	onto	seven	they	view	yours
might	opposite	seventh	third	wanting	yourself
mine	or	several	this	was	yourselves
minority	other	shall	those	we	
minus	ought	she	though	were	
more	our	should	three	what	
most	ours	similar	through	whatever	
much	ourselves	since	throughout	when	
must	out	six	thru	whenever	
my	outside	sixth	thus	where	

D.2 Additional Information Gain Results

In Section 5.3, Table 5.2 lists the top ten features by information gain. On the following page, we provide an extended version of this table which lists the top thirty. Note that the features "frequency of `.`" and "frequency of S-." are not identical. The latter only counts periods as a sentence terminator, but the former also counts other usages of that character, such as the decimal point. A similar remark applies to the features "frequency of `,`" and "frequency of S-,".

Feature	**Information Gain in Bits**
Frequency of ’	1.09666
Number of characters	1.07729
Frequency of words with only first letter uppercase	1.07275
Number of words	1.06049
Frequency of NP-PRP (noun phrase containing a personal pronoun)	1.03198
Frequency of .	1.02163
Frequency of all-lowercase words	1.01787
Frequency of NP-NNP (noun phrase containing a singular proper noun)	1.00869
Frequency of all-uppercase words	0.99070
Frequency of ,	0.94705
Frequency of S-. (sentence ending with a period)	0.94122
Frequency of ROOT-S (top-level clause is a declarative sentence)	0.91683
Frequency of one-character words	0.88404
Frequency of S-, (sentence containing a comma)	0.88381
Frequency of NP-PP (noun phrase containing a prepositional phrase)	0.87979
Frequency of ten-character words	0.87763
Frequency of S-VP (sentence containing a verb phrase)	0.87529
Frequency of –	0.87469
Frequency of S-NP (sentence containing a noun phrase)	0.87010
Frequency of nine-character words	0.86612
Frequency of PP-NP (prepositional phrase containing a noun phrase)	0.86540
Frequency of NP-NP (noun phrase containing a noun phrase)	0.86343
Frequency of words occurring once in a post (*hapax legomena*)	0.85528
Frequency of S-CC (sentence containing a coordinating conjunction)	0.85402
Frequency of VP-VBP (verb phrase containing a present tense verb, not third person singular)	0.85394
Frequency of PP-IN (prepositional phrase containing a subordinating preposition or conjunction)	0.85338
Frequency of S-S (sentence containing an independent clause, i.e., a compound sentence)	0.85133
Frequency of ADVP-RB (adverb phrase containing an adverb, not comparative or superlative)	0.85099
Frequency of eleven-character words	0.85038
Frequency of the word “my”	0.84940