Improved Magic Ink Signatures Using Hints

Markus Jakobsson[1] and Joy Müller[2]

[1] Information Sciences Research Center, Bell Laboratories
Murray Hill, NJ 07974
`markusj@research.bell-labs.com`
[2] Johannes Gutenberg Universität Mainz, Fachbereich Mathematik
55128 Mainz
`joy@dialup.nacamar.de`

Abstract. We introduce two improvements to the recently proposed so called *magic ink DSS signatures*. A *first* improvement is that we reduce the overhead for tracing without noticeably increasing any other cost. The tracing cost is linear in the number of generated signatures in the original proposal; our improved version reduces this to a *logarithmic* cost in the common case. A *second* improvement is that we introduce a method for determining whether forged currency is in circulation, without affecting the privacy of honest users.

Our improvements rely on our introducing a so called *hint value*. This is an encryption of the signature transcript received, submitted by the signature receiver. Part of the processing of this hint value is done using a new technique in which the high costs of secret sharing and robust computation on shared data are avoided by manipulation of encrypted data rather than plaintext. (Whereas the idea of computing on encrypted data is not a new notion in itself, it has to the best of our knowledge not previously been employed to limit the use of costly secret sharing based protocols.)

Keywords: efficiently revokable privacy, magic ink DSS, hints, electronic commerce

1 Introduction

Many changes in society are caused by the introduction of vital technology. An example of this is the invention of the printing press in 1457, which by a significant reduction in the costs of printed material caused a drastically increased literacy, and political awareness by allowing inexpensive information dissemination to the masses. Another example of an important step forward is the telephone, invented in 1876. These and innumerable other inventions caused and fueled the industrial revolution, transforming society in an eye-blink of human history. Although hardly anticipated only a few years ago, the Internet now promises to be a similar catalyst of changes to society. An integral part of this new revolution we are facing relates to commerce.

This new technology, however, promises to be as dangerous as it is useful. On the one hand, we know that given the strikingly low costs of information collection and analysis of the same, payment schemes not offering sufficient privacy may indeed act as an ever-present secret police in the hands of dubious commercial interests. On the other hand, it is also well-known that too *much* privacy may present a threat to society at large, in the form of terrorism, blackmailing, and the undermining of entire economies by corruption of or dissemination of the secret keys of banks (see [33,21]). A recent trend in research in the area of electronic commerce has therefore been to find payment schemes with revokable anonymity (e.g., [5,6,15,18,19,21,22,23,24,26,34,35,36],) allowing a set of trustees to remove the privacy of a given user or payment, but not allowing an attacker to correlate payments to the identities of the payers without corrupting a substantial number of these trustees.

Most signature schemes with revokable anonymity offer two types of tracing, namely: (1) from a given signing session or identity of a receiver to a description of the signature(s) obtained, and the opposite direction: (2) from a given signature to the corresponding signing session or identity of the signature receiver. Magic ink signatures [23] offer a third tracing option, which is to determine if a given signature was obtained in a given session or not. This third type allows a tighter control over tracing by allowing suspicions to be verified, without divulging any more information than whether the signature and the session match or not.

In the magic ink proposal in [23], the first and third tracing options have costs that are independent of the number of signatures that have been generated. The second tracing option, however, has an expected cost which is linear in the number of generated signatures. This is a concern in a practical implementation, especially given that this type of tracing is likely to be the most commonly needed.

A *first result* of this paper is to present a modification of the original magic ink scheme that lowers the cost of this second tracing option to a logarithmic cost [1] in the common case, with a fall-back to a linear cost in a highly unlikely case. This is done without affecting the other tracing costs, and with only a minor increase in storage costs for the signature generating servers.

A second issue we deal with relates to increasing the protection against attacks. One of the main benefits of magic ink signatures compared to other schemes with revokable anonymity is that it allows the signer/bank to distinguish between valid signatures that were produced by the bank servers, and valid signatures that were produced by *another* party holding the signing keys. This is important if there is a suspicion that the signing keys of the banks have been corrupted (corresponding to the so-called *bank robbery attack*). Whereas the availability of this method promises to act as a definitive deterrent against attacks aiming to corrupt the bank keys, the very high cost of the filtering makes

---

[1] This is using a naive search algorithm. Using a more efficient algorithm in which space is traded off for efficiency, a constant cost can be obtained.

the method impractical unless it is *certain* that the signing keys have been corrupted.

A *second result* of our paper is to answer the important question of how it can be detected at a very early stage that a bank key has been compromised. In previous proposals that are granting privacy to honest users, no such method was available. The crux of the idea is to detect valid bank signatures that have not been produced by the bank without affecting our stringent requirements on the privacy of honest users. This is similar in spirit to the notion of fail-stop signatures [31], which were proposed by Pfitzmann to allow the detection of valid but forged signatures. We propose a method for detecting forged DSA signatures, using the very same constructions introduced to lower the cost of tracing. This allows the above mentioned expensive filtering techniques to be employed only when necessary.

As a result, we obtain a highly efficient and practical signature generation scheme offering the following four important mechanisms: (1) tracing from a given signing session to the corresponding signature (or coin); (2) tracing from a given signature/coin to the corresponding signing session; (3) comparison of sessions and signatures to determine whether such a pair correspond to each other; and (4) detection of signing keys having been compromised. This last feature allows instantaneous installation of new secret and public keys, and an on-line filtering of all deposited coins of the old type to remove forged coins.

Thus, this new scheme, which offers improvements both in terms of efficiency and functionality, would potentially allow a realistic payment scheme that succeeds in balancing the scales of privacy in a way that avoids all known attacks and weaknesses.

Outline

Section 2 presents our general model. In section 3, we define the properties our scheme achieves. In section 4 we explain the tools we utilize to achieve our solution, which is discussed in section 5. This is followed in section 6 by the introduction of the main protocol for improved magic ink signatures and all sub-protocols that are needed. Section 7 presents the second result of our paper, namely a protocol for detecting illicit signatures. In section 8 we enumerate the properties of our scheme; these are proven in the Appendix.

2   Model

We assume that there are three types of (polynomial-time) participants: *signers/tracers* $\mathcal{S}$, *receivers* $\mathcal{R}$, and *verifiers*. A signer/tracer is an entity with two functionalities (as indicated by the name). When acting like a signer, this entity produces signatures on messages provided by the *receiver*; when acting like a tracer, it selectively correlates signatures and sessions that match (this will be elaborated on later.) The *receiver* sends message-signature pairs to a *verifier*, who wants to verify their validity with respect to the public key of the signer.

(Typically, these three entities correspond to banks, payers, and merchants; or, alternatively, to certification agencies, service providers, and users who want to determine whether a given service provider is certified.)

We consider two types of *adversaries*. The first, the so-called *mobile adversary* (introduced in [29]), may control up to a threshold of *signers/tracers*, and any number of *receivers* and *verifiers*. He can adaptively corrupt different signer/tracers for each time period, whose length is a security parameter set by the protocol designers. Our second type of adversary is an adversary of the first type who also has complete read access to the private storage areas of *all* signer/tracer servers. We call this adversary, which was introduced in [23], the *read-all adversary*.

3   Definitions

Terminology: We say that the predicate $match(s_i, \tau_i)$ is true if and only if s_i is the transcript the receiver obtains in the signing session i, and τ_i is the transcript obtained by the signer during the same session i.

Definition 1: Anonymity and Revokable Anonymity

Let $\mathcal{R}$ be a set of honest signature receivers and $\mathcal{S}$ is a set of honest signature servers. Additionally, let $\mathcal{R}'$ be a set of dishonest signature receivers, and $\mathcal{S}'$ a set of dishonest signature servers. We let the receivers in $\mathcal{R} \cup \mathcal{R}'$ interact in the proposed protocols with quorums (i.e., sets of sufficient size to recostruct the related secret) from $\mathcal{S}' \cup \mathcal{S}$ a polynomial number of times n, after which a receiver $R_i \in \mathcal{R}$ obtains a signature s_i on a message m_i of his choice. We assume that S' obtains the *set* of all generated signatures $\{s_i\}$, and a *list* of all signer-side transcripts $(\tau_1, \ldots, \tau_n)$.

We say our protocols implement **anonymity** if it is not possible for $\mathcal{R}' \cup \mathcal{S}'$ to match any signature s_i, obtained by a receiver in $\mathcal{R}$, to its corresponding signer-side transcript τ_i with probability better than what is achieved by making a guess uniformly at random from all transcripts produced during sessions with $\mathcal{R}$.

We say our protocols implement **revokable anonymity** if any quorum of honest servers $\mathcal{S}$ or tracing servers can perform the following three transactions in polynomial time:

1. Given a valid message-signature pair described by s_i and the list of all signer-side transcripts $(\tau_1, \ldots, \tau_n)$, select the value τ_j, such that $match(s_i, \tau_j)$.
2. Given a valid message-signature pair described by s_i and one signer-side transcript τ_j, determine whether $match(s_i, \tau_j)$ holds.
3. Given a signer-side transcript τ_i, compute a value $trace_i$ such that given $trace_i$ and a value s_j, a third party can determine in polynomial time, and without any interaction, whether $match(s_j, \tau_i)$ holds.

Definition 2: Unforgeability

As before, we let $\mathcal{R}$ be the set of honest signature receivers; $\mathcal{R}'$ the set of dishonest signature receivers; $\mathcal{S}$ the set of honest signature servers; and $\mathcal{S}'$ the set of dishonest signature servers. We let the receivers in $\mathcal{R} \cup \mathcal{R}'$ interact with quorums from $\mathcal{S}' \cup \mathcal{S}$ a polynomial number of times. Let σ be the set of valid message-signature pairs obtained by $\mathcal{R}$, and σ' the set of valid message-signature pairs obtained by $\mathcal{R}'$. We say that a signature[2] is *unforgeable* if it is infeasible for the adversary to output a valid message-signature pair that is not in $\sigma \cup \sigma'$.

Definition 3: Illicit Signature Detection

We refer to an *illicit* message-signature pair as any *valid* message-signature pair s for which there is no signer-side transcript τ such that $match(s, \tau)$. (That is, illicit signatures are valid signatures produced by an adversary who has corrupted the secret signing key used, or has broken the computational assumption of the signature scheme.) We call a system *illicit signature detecting* if it allows the signer/tracer to distinguish such an illicit message-signature pair from a message-signature pair that is not illicit, but which is produced by the signer.

4   Building Blocks

Before we introduce the improved version we review some protocols, which will later be used as building blocks.

Notation: Since we use different moduli at different times, we use $[op]_z$ to denote the operation op modulo z where this is not clear from the context.

ElGamal: Our protocol uses ElGamal encryption [16]. To encrypt a value m using the public key y, the person who performs the encryption picks a value $\gamma \in_u Z_q$ uniformly at random, and computes the pair $(a, b) = (my^\gamma, g^\gamma)$. Thus, (a, b) is the encryption of m. In order to decrypt this and obtain m, $m = a/b^x$ is calculated.

Mix-Networks: Consider an input list $(\alpha_1, \ldots, \alpha_n)$. A mix-network produces an output which is a random (and secret) permutation of $(\alpha_1{}^x, \ldots \alpha_n{}^x)$, for a given secret key $x \in Z_q$.
We will use a robust (i.e., such that it produces the correct output given an honest quorum of participants) mix-network [10] decryption scheme, such as [1,25,27].

---

[2] This refers both to the transcript and the method of generating the transcript.

The Digital Signature Standard (DSS): We use the DSS [7] (described herein) as the underlying signature algorithm.

Key Generation. A DSS key is composed of public information p, q, g, a public key y and a secret key x, where:
1. p is a prime number of length l where l is a multiple of 64 and $512 \leq l \leq 1024$.
2. q is a 160-bit prime divisor of $p - 1$.
3. g is an element of order q in Z_p^*.
4. x is the secret key of the signer, a random number $1 \leq x < q$.
5. $y = [g^x]_p$ is the public verification key.

Signature Algorithm. Let $m \in Z_q$ be a hash of the message to be signed. The signer picks a random number k such that $1 \leq k < q$, calculates $k^{-1} \bmod q$ (w.l.o.g. k and k^{-1} values compared to DSA description are interchanged), and sets

$$r = [[g^{k^{-1}}]_p]_q$$
$$s = [k(m + xr)]_q$$

The pair (r, s) is a signature of m.

Verification Algorithm. A signature (r, s) of a message m can be publicly verified by checking that $r = [[g^{ms^{-1}} y^{rs^{-1}}]_p]_q$.

Magic Ink Signatures

As the underlying signature algorithm we use the DSS. For simplicity, we only show a single-server method for producing Magic Ink DSS signatures. Since the privacy depends on the distribution of the signer, the latter must be distributed. The real generation and tracing protocols are therefore distributed variants of this protocol, in order to increase the availability and security of the system and to introduce control (see [23]).

1. S generates a random secret session key, $\overline{k} \in_u Z_q$, and computes $\overline{r} = [g^{\overline{k}^{-1}}]_p$.
2. The signature receiver R has a hashed message $m \in Z_q$ that he wants signed. He generates two blinding factors, $\alpha, \beta \in_u Z_q$ and computes $r = [[\overline{r}^{\beta}]_p]_q$, $\mu = [m\alpha]_q$. and $\rho = [r\alpha]_q$.
   R sends (μ, ρ) to the signature generating server S.
3. S produces a tag, which will be a function of the signature transcript, and which uniquely identifies this. (We describe this step in more detail later). This *tag* is used for tracing, in case of anonymity revocation.
   Then, S generates the DSS signature $\sigma = [\overline{k}(\mu + x\rho)]_q$ on the message μ, using the blinded public session key ρ. The server sends σ to R.
4. The signature receiver R unblinds the signature: $s = [\sigma\alpha^{-1}\beta^{-1}]_q$. The triple (m, r, s) is a valid DSS signature on m.

Tracing: There are three types of tracing we can perform:

1. **From known signing session to signed message:** The signature invariant is calculated from the tag of a given session. This signature invariant uniquely identifies the corresponding signature.
2. **From known signed message to signing session:** Given a signature, we wish to find the corresponding signing session. In the first magic ink proposal [21], this was done by computing a trace value that was compared to the tag of each potential signing session. In this paper we suggest a more efficient method for this type of tracing.
3. **By comparison:** The signature invariant of a given signature is compared to the tag of a given session. The output is one bit, whether they correspond to each other or not.

5   Our Solution

Consider the tracing type 2 above, that is, from a given signature to the corresponding signing session. The idea is to introduce values that are voluntarily submitted by the receiver and that can be used to very efficiently trace from a signature to a signing session. The reason we say that these values, which we call *hints*, are *voluntarily submitted* is that due to efficiency requirements, there are no controls on their correctness. It makes little sense for a user to submit an incorrect value, however, since the difference will just be whether the tracing (if needed) will require a low or a medium amount of computational resources. Also, even though an incorrect value will not be detected during the signing (withdrawal) process, it will be detected by the mechanism for illicit signature generation, after which the signature/coin can be traced and revoked, and the user punished. Efficiently, this will make the cost for tracing logarithmic, even though we rely on a fall-back to the linear-time tracing mechanism of the old magic ink solution if the wrong hint value is submitted.

A hint can be thought of as an encryption of the signature the receiver obtains, with the property that it is not possible for an adversary corrupting less than a quorum of signer/tracer servers to compute the hint value corresponding to a signature (and vice versa), while a quorum of signer/tracers can efficiently compute the hint value given a signature. The signing process gives the receiver the signature on a message, and gives the signers, among other things, the hint value, which is stored along with other tracing values and the identity of the receiver.

In order to trace from a signature to a signature session, a quorum of tracing servers compute the hint value from the signature, and select the corresponding record (from a list of sessions that has indices sorted with respect to hint values). If no record is found, we use a fall-back to the linear search method described in [23].

Our signature and hint generation method involves a proof of knowledge by the receiver, to guarantee that the hint value submitted is not a function of hints submitted in previous sessions. (If this were not checked then it would

potentially undermine the privacy of other users.) We prove that the addition of the hint value, the process to obtain it, and the other new additions to the scheme do not negatively affect any desired protocol property.

The traditional way to distributively compute and verify the correct form of a value such as the hint value involves sharing of the secret value submitted by the receiver. This is found, however, to drastically increase the costs of the proofs of partial correctness, and we instead take another approach, namely to perform a computation on an encrypted transcript. Although this method is not conceptually novel, it has been used only to a very limited extent, mostly due to the difficulties of computing on encrypted data. We find that the type of computation we need to perform here for the processing of the hint value can be done very efficiently on encrypted data. This method might therefore be of independent interest, and might be applied to similar situations in order to boost the efficiency of other multi-party computations.

Note also that this new method does not affect in any way the resulting signature: the signature obtained by the receiver is still a standard DSS signature (on a message of a particular format.) We believe that this is an important point in order to allow commerial use of the scheme, and to benefit from the careful scrutiny of the DSS that has been performed. The only negative side-effect of our new signature generation scheme, as far as we can see, is the nominal increase in communication and computation of the parties involved, and the small increase in the size of database that is kept by the signer/tracer.

6   Improved Distributed Magic Ink Signatures

Let us now consider a distributed version of the protocols previously presented. Here, let Q be a quorum of t servers in $S_1 \ldots S_n$. We assume that the message m to be signed is of the form $m = f^M \bmod p$ for a generator f. Commonly, this type of scheme is used to sign a public key, in which m is this public key, and M is its corresponding secret key. (For messages M that can be guessed with a non-negligible probability, an alternative form $m = f_1{}^M f_2{}^R$ for a random R can be employed.)

System initialization: The servers distributively generate a random secret x for signature generation, using a (t_s, n) secret sharing scheme, a random secret x_t for tracing, using a (t_t, n) secret sharing scheme, and a random secret x_h for hint generation, using a (t_h, n) secret sharing scheme. Each server S_i publishes his shares of the public keys $y_i = [g^{x_i}]_p$, $y_{ti} = [g^{x_{ti}}]_p$, and $y_{hi} = [g^{x_{hi}}]_p$, from which $y = [g^x]_p$, $y_t = [g^{x_t}]_p$ and $y_h = [g^{x_h}]_p$ are interpolated (we refer to [30] for a discussion of how this is done.) Each server then proves knowledge of his secret shares x_i, x_{ti} and x_{hi} to the other servers; if some server fails, then he is replaced and the protocol restarts. Finally, the signing public key y is published.

Session initialization: Before starting the signature generation protocol, the receiver R has to send his identity id and a proof of knowledge of the se-

cret key corresponding to id. The signers pick a session identification number, $sessionid = id||l$, where l is a number making $sessionid$ a unique string.

Signature generation:

1. The servers prepare a temporary key pair:
   (a) The set of servers $S_i|i \in Q$ distributively generate the private session key $\overline{k} \in_u Z_q$.
   (b) Server S_i has a share $\overline{k}_i$ and publishes $[g^{\overline{k}_i}]_p$.
   (c) The servers compute $\overline{r} = [g^{\overline{k}^{-1}}]_p$, using the methods for computing reciprocals in [20].
   (d) $\overline{r}$ is sent to the signature receiver R.
2. The receiver R wants a signature on the message $m = [h^M]_p$.
   (a) He generates two blinding factors, $\alpha, \beta \in_u Z_q$.
   (b) He computes blinded versions of m and $\overline{r}$: $\mu = [m\alpha]_q$, $r = [[\overline{r}^\beta]_p]_q$ and $\rho = [r\alpha]_q$.
   (c) Using a (t_s, n) secret sharing, he computes $(\mu_1, \ldots \mu_n)$ of μ, with public information $(g^{\mu_1} \ldots g^{\mu_n})$ and a (t_s, n) secret sharing $(\rho_1, \ldots \rho_n)$ of ρ, with public information $(y_t^{\rho_1} \ldots y_t^{\rho_n})$.
   (d) He computes an ElGamal encryption of m w.r.t. the public hint key y_h: $(a, b) = (mg^\gamma, y_h^\gamma)$., where $\gamma \in Z_q$.
   (e) He sends (μ_i, ρ_i, a, b) to signature generating server S_i.

3. The tracing values and the signature are generated.
   (a) The servers interpolate the tag, $tag = ([g^\mu]_q, [y_t^\rho]_q)$.
   (b) After they have verified the correctness of the computation of (a, b) (for which we present a robust protocol below), they robustly calculate the hint value $hint = a^{x_h}/b$. If R did not cheat, this value equals m^{x_h}.
   (c) The $hint$ is stored in a record along with tag, $sessionid$ and id.
   (d) The set of servers $S_i|i \in Q$ distributively generate the DSS signature σ on the message μ, using the (shared) public session key ρ; σ is calculated as follows: S_i generates $\sigma_i = [\overline{k}_i(\mu_i + x_i\rho_i)]_q$. Then, $\sigma = [\overline{k}(\mu + x\rho)]_q$ is interpolated from the σ_i's using the method for multiplication of secrets in [20].
   (e) The servers send σ to R.

4. The signature receiver R unblinds the signature: $s = [\sigma\alpha^{-1}\beta^{-1}]_q$. The triple (m, r, s) is a valid DSS signature on m.

Hint-generation:

Let x_h be a private key distributively held by the tracing servers; $y_h = [g^{x_h}]_p$ is the corresponding public key.

1. The receiver calculates an ElGamal encryption of m: He chooses a $\gamma \in_u Z_q$ and calculates $(a, b) = (mg^\gamma, y_h^\gamma) = (f^M g^\gamma, y_h^\gamma)$. This pair is sent to the servers.

2. (a) The servers distributively compute $hint_i = a^{x_{hi}}/b$.
   (b) In order to prove that every server has performed the correct exponentiation the servers run a protocol for proving valid exponentiation, e.g., [12,32]. This is a proof that $log_a(hint_i b) = log_g(y_{hi})$ for a given quadruple $(a, g, (hint_i b), y_{hi})$.
   (c) The servers compute $hint$ as the Lagrange-weighted product of the shares $hint_i$ of the servers in the quorum. (This value equals $[m^{x_h}]_p$ if R did not cheat.)

We also have to force the receiver to prove that a is computed the right way. This can be done using the method described below:

Avoiding Subliminal Tracing

Attacks are possible if it is possible for an attacker to inject previously seen encryptions, or functions of these, and observe what hint is produced. The potential problem is if an attacker would use the hint-generation protocol as an oracle to compute a hint of a seen signature. For example, assume an attacker could take a value m' of a signature he has seen "on the street", encrypt this (claiming to withdraw a new coin) and send $(a, b) = (m'g^\gamma, y_h^\gamma)$ to the servers. Then one dishonest server would watch to see what value $hint = m'^{x_h}$ is produced: this efficiently traces the value m', because the dishonest participants get to know the corresponding record of the signature. Therefore, the user has to prove that he knows the format of the portion of the encryption that will be raised to the x_h power. If he knows a representation, it cannot be a signature "on the street".

   Our solution for the encryption need to satisfy plaintext awareness (The best description of this concept is probably that of Bellare, Desai, Pointcheval and Rogaway [2]). This guarantees that the receiver knows the plaintext, preventing this attack. Note, though that this must be done without revealing any transcript-specific information.

   We do it by proving knowledge that $(a, b) = (f^M g^\gamma, y_h^\gamma)$, without leaking any information about the message $m = f^M$. As mentioned above, we are only concerned about the value a; if b isn't of the right form that only would give us a wrong hint-value.

Since the servers have to verify the computation of a in step 3b of the signature generation protocol, the receiver has to prove knowledge during step 2 and step 3a.

1. Each verifier S_i, $i \in Q$ (which in our case corresponds to a participating signing server) selects a value $\epsilon_i \in_u Z_q$. S_i publishes $(\hat{f}_i, \hat{g}_i) = ([f^{\epsilon_i}]_p, [g^{\epsilon_i}]_p)$. The pair $(\hat{f}, \hat{g}) = ([\prod_{i \in Q} \hat{f}_i]_p, [\prod_{i \in Q} \hat{g}_i]_p)$ is sent to the signature receiver.

2. The prover (in our case, the signature receiver) computes $\hat{a} = [\hat{f}^M \hat{g}^\gamma]_p$, where M is the preimage of m and γ is the blinding exponent chosen for the ElGamal encryption. The prover sends a commitment $com(\hat{a})$ to the verifiers.

3. Each verifier S_i publishes his value ϵ_i, and $\epsilon = \left[\sum_{i \in Q} \epsilon_i \right]_q$ is sent to the prover.
4. The prover verifies that $(\hat{f}, \hat{g}) = ([f^\epsilon]_p, [g^\epsilon]_p)$ and halts if this is not satisfied. Otherwise, he decommits to his commitment of $\hat{a}$ to the verifiers.
5. Each verifier checks that $\hat{a} = [a^\epsilon]_p$, and accept iff this holds.

Tracing

We recall that we have a secret key x for signing, a secret key x_t for tracing, and a secret key x_h for generating the hint. Furthermore we have $tag = ([g^\mu]_q, [y_t^\rho]_q)$. The three types of tracing are performed as follows:

1. **From known signing session to signed message:** *(unchanged)*
   The pair $(trace_a, trace_b) = (tag_a^{x_t}, tag_b)$ is calculated by any size-$(t_t + 1)$ quorum of holders of shares of x_t . This pair is output. A certain signature, described by (m, r, s), corresponds to the given tag if $trace_a^{rm^{-1}} \equiv_p trace_b$.
2. **From known signed message to signing session:**
   Given a description (m, r, s), the tracing servers compute a value $trace_c = [m^{x_h}]_p$. Then they compare $trace_c$ with the stored hints.
   If $trace_c \equiv_p hint$ for a particular record, then the signed message corresponds to the signing session of this record.
   If there is no such $hint$ which equals $trace_c$, then the tracing servers have to calculate $(trace_a, trace_b) = ([tag_a^{rm^{-1}}]_p, tag_b)$ for each potential withdrawal session. Using a protocol for verification of undeniable signatures [12], they verify whether $log_g(y_t) = log_{trace_a} trace_b$, which holds if the signature corresponds to the tag.
3. **By comparison:** *(unchanged)*
   Given is a $tag = (g^\mu, y_t^\rho)$ and a signature (m, r, s). The tracing servers calculate $(trace_a, trace_b) = ([tag_a^{rm^{-1}}]_p, tag_b)$. Using the protocol for verification of undeniable signatures, we verify whether $log_g(y_t) = log_{trace_a} trace_b$, which holds if the signature corresponds to the tag.

7   Illicit Signature Detection

This section briefly presents our second result in this paper, which is a method to detect that the secret signing key has been compromised.
We let the signers periodically blind all the hints for valid sessions, and, using a mix-network, blind portions of the recently "deposited" signatures, and then verify that each blinded deposited transcript corresponds to a blinded session transcript. If there is any blinded deposited transcript that has no match, then this is unblinded and traced. If, during tracing, a matching session is not found, then the servers output "signing key compromised" as this signature cannot have been produced by the signature servers. Otherwise, the signature simply had an incorrect hint value submitted, in which case appropriate action is taken to punish the withdrawer.

We will now present the protocol in more detail:

1. As input the mix servers have a list $(hint_1, \ldots, hint_K)$, which have been generated during signature generation protocols. A blinding exponent ζ is distributively chosen so that $\zeta = \prod_{i \in Q} \zeta_i$, where ζ_i is the share held by server S_i. The servers robustly compute $(hint_1^{\zeta}, \ldots, hint_K^{\zeta})$.
2. (a) The servers have a list $(m_1, \ldots, m_k)$ corresponding to the messages of all the recently deposited signatures (i.e., those deposited since the last run of the detection protocol.)
   (b) They robustly blind this list with the same blinding exponent ζ and get $(m_1^{\zeta}, \ldots, m_k^{\zeta})$.
   (c) The mix servers perform a *mix-decryption* on this list, resulting in a permutation of the list $(\hat{hint}_1, \ldots, \hat{hint}_k)$, where $\hat{hint}_i = \left(m_i^{\zeta}\right)^{x_h}$.
3. All entries from the second list that exist as entries in the first list are removed. Each remaining item $\hat{hint}_i$ is unblinded by computing $m_i = \hat{hint}_i^{1/(\zeta x_h)}$. Each corresponding signature is traced using standard methods (see section 6). If the trace is successful, the receiver of the signature is punished for having given the incorrect hint value; if there is an unsuccessful trace, then the servers output "signing key corrupted".

8   Claims

We claim that our scheme achieves anonymity (Theorem 1), revokable anonymity (Theorem 2), unforgeability (Theorem 3) and illicit signature detection (Theorem 4). The theorems are sketched in the appendix. A full version of the proofs, omitted due to space limitations, is available from the authors upon request, and will be part of the second author's Master's thesis.

Acknowledgements

Many thanks to Ari Juels and Julien Stern for helpful feedback.

References

1. M. Abe, "Universally Verifiable Mix-net with Verification Work Independent of the Number of Mix-servers," Advances in Cryptology - Proceedings of Eurorypt '98, pp. 437–447.   257
2. M. Bellare, A. Desai, D. Pointcheval, P. Rogaway, "Relations Among Notions of Security for Public-Key Encryption Schemes," Advances in Cryptology - Proceedings of Crypto '98, pp. 26–45.   262
3. S. Brands, "Untraceable Off-line Cash in Wallets with Observers," Advances in Cryptology - Proceedings of Crypto '93, pp. 302–318.
4. S. Brands, "An Efficient Off-line Electronic Cash Systems Based on the Representation Problem," C.W.I. Technical Report CS-T9323, The Netherlands.

5. E. Brickell, P. Gemmell and D. Kravitz, "Trustee-based Tracing Extensions to Anonymous Cash and the Making of Anonymous Change," Proc. 6th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 1995, pp. 457–466. 254

6. J. Camenisch, U. Maurer and M. Stadler, "Digital Payment Systems with Passive Anonymity-Revoking Trustees," Computer Security - ESORICS 96, volume 1146, pp. 33–43. 254

7. National Institute for Standards and Technology, "Digital Signature Standard (DSS)," Federal Register Vol 56(169), Aug 30, 1991. 258

8. J. Camenisch, J-M. Piveteau and M. Stadler, "An Efficient Fair Payment System," Proceedings of the 3rd ACM Conference on Computer and Communications Security, 1996, pp. 88–94.

9. D. Chaum, A. Fiat and M. Naor, "Untraceable Electronic Cash," Advances in Cryptology - Proceedings of Crypto '88, pp. 319–327.

10. D. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," Communications of the ACM, ACM 1981, pp. 84-88. 257

11. D. Chaum, "Blind Signatures for Untraceable Payments," Advances in Cryptology - Proceedings of Crypto '82, pp. 199-203.

12. D. Chaum, H. Van Antwerpen, "Undeniable Signatures," Advances in Cryptology - Proceedings of Crypto '89, pp. 212-216. 262, 263

13. D. Chaum, "Achieving Electronic Privacy," Scientific American, August 1992, pp. 96–101.

14. D. Chaum and T. Pedersen, "Wallet databases with observers," Advances in Cryptology - Proceedings of Crypto '92, pp. 89–105.

15. G.I. Davida, Y. Frankel, Y. Tsiounis, and M. Yung, "Anonymity Control in E-Cash Systems," Financial Cryptography 97, pp. 1–16. 254

16. T. ElGamal, "A Public-Key Crytosystem and a Signature Scheme Based on the Discrete Logarithmus," Advances in Cryptology - Proceedings of Crypto '84, pp. 10-18. 257

17. N. Ferguson, "Extensions of Single-term Coins," Advances in Cryptology - Proceedings of Crypto '93, pp. 292–301.

18. Y. Frankel, Y. Tsiounis, and M. Yung, "Indirect Discourse Proofs: Achieving Efficient Fair Off-Line E-Cash," Advances in Cryptology - Proceedings of Asiacrypt 96, pp. 286–300. 254

19. E. Fujisaki, T. Okamoto, "Practical Escrow Cash System", LNCS 1189, Proceedings of 1996 Cambridge Workshop on Security Protocols, Springer Verlag, pp. 33 – 48. 254

20. R. Gennaro, S. Jarecki, H. Krawczyk, T. Rabin, "Robust Threshold DSS Signatures", Advances in Cryptology - Proceedings of Eurocrypt '96, pp. 354-371. 261, 261

21. M. Jakobsson and M. Yung, "Revokable and Versatile Electronic Money," 3rd ACM Conference on Computer and Communications Security, 1996, pp. 76–87. 254, 254, 259

22. M. Jakobsson, "Privacy vs Authenticity," PhD Thesis, University of California, San Diego, 1997. 254

23. M. Jakobsson and M. Yung, "Distributed 'Magic Ink' Signatures," Advances in Cryptology - Proceedings of Eurocrypt '97, pp. 450–464. 254, 254, 254, 256, 258, 259, 267

24. M. Jakobsson and M. Yung, "Applying Anti-Trust Policies to Increase Trust in a Versatile E-Money System," Advances in Cryptology - Proceedings of Financial Cryptography '97, pp. 217–238. 254

25. M. Jakobsson, "A Practical Mix," Advances in Cryptology - Proceedings of Eurocrypt '98, pp. 448-461. 257
26. D. M'Raïhi, "Cost-Effective Payment Schemes with Privacy Regulation," Advances in Cryptology - Proceedings of Asiacrypt '96. 254
27. W. Ogata, K. Kurosawa, K. Sako, K. Takatani, "Fault Tolerant Anonymous Channel," ICISC '97, pp. 440–444. 257
28. T. Okamoto, "An Efficient Divisible Electronic Cash Scheme," Advances in Cryptology - Proceedings of Crypto '95, pp. 438–451.
29. R. Ostrovsky and M. Yung,"How to withstand mobile virus attacks," Proc. of the 10th ACM Symposium on the Principles of Distributed Computing, 1991, pp. 221-242. 256
30. T.P. Pedersen, "Distributed Provers with Applications to Undeniable Signatures," Advances in Cryptology - Proceedings of Eurocrypt '91, pp. 221-242. 260
31. Birgit Pfitzmann, "Digital Signatures Schemes-General Framework and Fail-Stop Signatures," LLNC1100, Springer-Verlag, Berlin 1996 255
32. C.P. Schnorr, "Efficient Signature Generation for Smart Cards," Advances of Cryptology, Proceedings of Crypto '98, pp.239–252. 262
33. S. von Solms and D. Naccache, "On Blind Signatures and Perfect Crimes," Computers and Security, 11 (1992) pp. 581–583. 254
34. M. Stadler, "Cryptographic Protocols for Revokable Privacy," PhD Thesis, ETH No. 11651, Swiss Federal Institute of Technology, Zürich, 1996. 254
35. M. Stadler, J-M. Piveteau and J. Camenisch, "Fair Blind Signatures," Advances in Cryptology - Proceedings of Eurocrypt '95, pp. 209–219. 254
36. Y. Tsiounis, "Efficient Electronic Cash: New Notions and Techniques," PhD Thesis, College of Computer Science, Northeastern University, 1997. `http://www.ccs.neu.edu/home/yiannis` 254
37. B. Witter, "The Dark Side of Digital Cash," Legal Times, January 30, 1995.

9   Appendix

Theorem 1: Let D denote the dishonest signature servers, and H the honest signature servers.

An adversary $\mathcal{A}$, who controls any set of less than or equal to t_t tracing dishonest servers S_j $j \in D$ and dishonest receiver R, cannot break the anonymity of the signature scheme, i.e., he is not able to determine whether $match(s, \tau)$ holds for a particular pair (s, τ) with probability non-negligibly better than a guess.

Outline of Proof of Theorem 1: Employing an oracle for generating valid DSS signatures, we provide a simulator $\mathcal{S}$ for all the protocols. Each simulation generates transcripts that cannot be distinguished from the real protocol transcripts by an adversary $\mathcal{A}$ as above. We then compose the individual simulations to form a simulator for the entire protocol. We show that the adversary $\mathcal{A}$ cannot distinguish the transcripts generated during the simulation from the transcripts generated by a real protocol. ¿From this we can conclude that the adversary cannot break the anonymity of the scheme. This must hold since the adversary can produce the same transcripts himself by the use of the simulator and the simulator does not have access to the secret key for tracing. □

Theorem 2: The system achieves revokable anonymity, i.e., any quorum of honest tracing servers is able to perform the following actions: (a) Given a valid message-signature pair described by s_i and the list of all signer-side transcripts $(\tau_1, \ldots, \tau_n)$, select the value τ_j, such that $match(s_i, \tau_j)$. (b) Given a valid message-signature pair described by s_i and one signer-side transcript τ_j, determine whether $match(s_i, \tau_j)$ holds. (c) Given a signer-side transcript τ_i, compute a value $trace_i$ such that given $trace_i$ and a value s_j, a third party can determine in polynomial time, and without any interaction, whether $match(s_j, \tau_i)$ holds.

We refer to [23] for the proof.

Theorem 3: The system achieves unforgeability, i.e., an adversary $\mathcal{A}$, who controls any set of less than or equal to t_t tracing dishonest servers S_j $j \in D$ and dishonest receiver R', cannot generate a signature that gets accepted as valid.

Outline of Proof of Theorem 3: Assume the contrary. Then it must be possible to construct a valid signature given only the shares of the secret signature generation key x held by the t_t dishonest servers. Since the secret key has been shared with a (t, n) threshold scheme, it is not possible to reconstruct the secret key with less the $t + 1$ shares. Therefore $\mathcal{A}$ only has shares that are statistically uncorrelated to the secret key, this would by a simulation argumentation imply that valid signatures could be generated without any secret knowledge. □

Theorem 4: The system is detecting illicit signatures, i.e., the signer/tracer is able to check whether there exist a signer-side transcript τ such that $match(s, \tau)$ holds.

Outline of Proof of Theorem 4: There are exactly three types of valid signatures: (1) those with *correct* hint values, (2) those with *incorrect* hint values, and (3) those with *no* hint values. For each signature the signature servers generate, a valid or invalid hint is produced (which one depends on whether the receiver is honest or not) and stored in the signer database, to which all writes are detected by the signers, and therefore, to which only the signers can write. When a signature is generated by the adversary, no hint is therefore stored in this database.

Each signature (m, r, s) with a *valid* hint can be matched to its singature session by computing $hint = m^{x_h}$, which can always be done by a quorum of servers. This value identifies the signing session. Each signature with an *invalid* hint can be matched to its signing session by finding a pair (tag_a, tag_b) in the signer database, such that $tag_a{}^{rm^{-1}} = tag_b$. This pair, which again identifies the session, is robustly computed during the signing session, and so, must exist in the database for valid signing sessions. An illicit signature has no session record stored. It is not possible to produce a valid signature (m, r, s) on a known preimage M such that this record matches a recorded hint value or tag value. The former holds since the hint value determines the message m, and therfore also M; the latter holds since a given triple (m, tag_a, tag_b) determines the value r, which would pre-

vent a valid signature to be produced (we refer to a description of DSS for this.) □