Full version of an extended abstract published in Proceedings of ACM CCS 2006, ACM Press, 2006. Available from the IACR Cryptology ePrint Archive as Report 2006/454.

How to Win the Clone Wars: Efficient Periodic n-Times Anonymous Authentication

Jan Camenisch* Susan Hohenberger† Markulf Kohlweiss‡ Anna Lysyanskaya§
Mira Meyerovich ¶

Abstract

We create a credential system that lets a user anonymously authenticate at most n times in a single time period. A user withdraws a dispenser of n e-tokens. She shows an e-token to a verifier to authenticate herself; each e-token can be used only once, however, the dispenser automatically refreshes every time period. The only prior solution to this problem, due to Damgård et al. [30], uses protocols that are a factor of k slower for the user and verifier, where k is the security parameter. Damgård et al. also only support one authentication per time period, while we support n. Because our construction is based on e-cash, we can use existing techniques to identify a cheating user, trace all of her e-tokens, and revoke her dispensers. We also offer a new anonymity service: glitch protection for basically honest users who (occasionally) reuse e-tokens. The verifier can always recognize a reused e-token; however, we preserve the anonymity of users who do not reuse e-tokens too often.

1 Introduction

As computer devices get smaller and less intrusive, it becomes possible to place them everywhere and use them to collect information about their environment. For example, with today's technology, sensors mounted on vehicles may report to a central traffic service which parts of the roads are treacherous, thus assisting people in planning their commutes. Some have proposed mounting sensors in refrigerators to report the consumption statistics of a household, thus aiding in public health studies, or even mounting them in people's bodies in an attempt to aid medical science. In all these areas, better information may ultimately lead to a better quality of life.

Yet this vision appears to be incompatible with privacy. A sensor installed in a particular car will divulge that car's location, while one installed in a fridge will report the eating and drinking habits of its owner.

*IBM Research, Zurich Research Laboratory, CH-8803 Rüschlikon, Switzerland, jca@zurich.ibm.com
†IBM Research, Zurich Research Laboratory, CH-8803 Rüschlikon, Switzerland, sus@zurich.ibm.com
‡Deptartment of Electrical Engineering, Katholieke Universiteit Leuven, B-3001 Heverlee, Belgium, mkohlwei@esat.kuleuven.be
§Computer Science Department, Brown University, Providence, RI 02912, USA, anna@cs.brown.edu
¶Computer Science Department, Brown University, Providence, RI 02912, USA, mira@cs.brown.edu

A naive solution would be to supply only the relevant information and nothing else.[1] A report about the road conditions should not say which sensor made the measurement. However, then nothing would stop a malicious party from supplying lots of false and misleading data. We need to authenticate the information reported by a sensor without divulging the sensor's identity. We also need a way to deal with rogue sensors, i.e., formerly honest sensors with valid cryptographic keys that are captured by a malicious adversary and used to send lots of misleading data.

The same problem arises in other scenarios. Consider an interactive computer game. Each player must have a license to participate, and prove this fact to an on-line authority every time she wishes to play. For privacy reasons, the player does not wish to reveal anything other than the fact that she has a license. How can we prevent a million users from playing the game for the price of just one license?

A suite of cryptographic primitives such as group signatures [28, 22, 1, 7] and anonymous credentials [26, 31, 40, 15, 17, 18] has been developed to let us prove that a piece of data comes from an authorized source without revealing the identity of that particular source. However, none of the results cited above provide a way to ensure anonymity and unlinkability of honest participants while at the same time guaranteeing that a rogue cannot undetectably provide misleading data in bulk. Indeed, it seems that the ability to provide false data is a consequence of anonymity.

Recently Damgård, Dupont and Pedersen [30] presented a scheme that overcomes this seeming paradox. The goal is to allow an honest participant to anonymously and unlinkably submit data at a small rate (for example, reporting on road conditions once every fifteen minutes, or joining one game session every half an hour), and at the same time to have a way to identify participants that submit data more frequently. This limits the amount of false information a rogue sensor can provide or the number of times that a given software license can be used per time period.

While the work of Damgård et al. is the first step in the right direction, their approach yields a prohibitively expensive solution. To authenticate itself, a sensor acts as a prover in a zero-knowledge (ZK) proof of knowledge of a relevant certificate. In their construction, the zero-knowledge property crucially depends on the fact that the prover must make some random choices; should the prover ever re-use the random choices he made, the prover's secrets can be efficiently computed from the two transcripts. The sensor's random choices are a pseudorandom function of the current time period (which must be proven in an additional ZK proof protocol). If a rogue sensor tries to submit more data in the same time period, he will have to use the same randomness in the proof, thus exposing his identity. It is very challenging to instantiate this solution with efficient building blocks. Damgård et al. use the most efficient building blocks available, and also introduce some of their own; their scheme requires that the user perform $57+68k$ exponentiations to authenticate, where k is the security parameter (a sensor can cheat with probability 2^{-k}).

We provide a completely different approach that yields a practical, efficient, and provably secure solution. We relate the problem to electronic cash (e-cash) [24, 25] and in particular, to compact e-cash [13]. In our approach, each participant obtains a set of e-tokens from the central server. Similar to the withdrawal protocol of e-cash, the protocol through which a participant obtains these e-tokens does not reveal any information to the server about what these e-tokens actually look like. Our protocol lets a participant obtain all the e-tokens it will ever need in its lifetime in one efficient transaction. The user performs only 3 multi-base exponentiations to obtain e-tokens,

[1]Note that divulging the relevant information alone may already constitute a breach of privacy; in this paper, we do not address this aspect of the problem; it has more to do with statistical properties of the data itself. See Sweeney [47] and Chawla et al. [29] on the challenges of determining which data is and is not safe to reveal.

and 35 multi-base exponentiations to show a single e-token. If the user is limited to one e-token per time period (as in the Damgård et al.'s scheme), the scheme can be further simplified and the user will need to do only 13 multi-base exponentiations to show an e-token. We provide more details on efficiency in §5.3.

Distributed sensors can use an e-token to anonymously authenticate the data they send to the central server. In the on-line game scenario, each e-token can be used to establish a new connection to the game. Unlike e-cash, where it is crucial to limit the amount of money withdrawn in each transaction, the number of e-tokens obtained by a participant is unlimited, and a participant can go on sending data or connecting to the game for as long as it needs. The e-tokens are anonymous and unlinkable to each other and to the protocol where they were obtained. However, the number of e-tokens that are valid during a particular time period *is* limited. Similarly to what happens in compact e-cash, reusing e-tokens leads to the identification of the rogue participant. We also show how to reveal all of its past and future transactions.

Thus, in the sensor scenario, a sensor cannot send more than a small number of data items per time period, so there is a limit to the amount of misleading data that a rogue sensor can submit. Should a rogue sensor attempt to do more, it will have to reuse some of its e-tokens, which will lead to the identification of itself and possibly all of its past and future transactions. Similarly, in the on-line game scenario, a license cannot be used more than a small number of times per day, and so it is impossible to share it widely.

OUR CONTRIBUTION Our main contribution is the new approach to the problem, described above, that is an order of magnitude more efficient than the solution of Damgård et al. In Section 5, we present our basic construction, which is based on previously-proposed complexity theoretic assumptions (SRSA and y-DDHI) and is secure in the plain model.

Our construction builds on prior work on anonymous credentials [15, 39], so that it is easy to see which parts need to be slightly modified, using standard techniques, to add additional features such as an anonymity revoking trustee, identity attributes, etc. The computational cost of these additional features is a few additional modular exponentiations per transaction.

In Section 6, we extend our basic solution to make it tolerate occasional glitches without disastrous consequences to the anonymity of a participant. Suppose that a sensor gets reset and does not realize that it has already sent in a measurement. This should not necessarily invalidate all of the sensor's data. It is sufficient for the data collection center to notice that it received two measurements from the same sensor, and act accordingly. It is, of course, desirable, that a sensor that has too many such glitches be discovered and replaced. Our solution allows us to be flexible in this respect, and tolerates m such glitches (where m is specified ahead of time as a system-wide parameter) at the additive cost of $O(km)$ in both efficiency and storage, where k is the security parameter. This does not add any extra computational or set-up assumptions to our basic scheme.

In Section 7, we consider more variations of our basic scheme. We show, also in the plain model, how to enable the issuer and verifiers to prove to third parties that a particular user has (excessively) reused e-tokens (this is called *weak exculpability*); and enable the issuer and verifiers to trace all e-tokens from the same dispenser as the one that was excessively reused (this is called *tracing*). We also show, in the common-parameters and random-oracle models, how to achieve *strong exculpability*, where the honest verifiers can prove to third parties that a user reused a particular e-token. Finally, we explain how e-token dispensers can be revoked; this requires a model where the revocation authority can continuously update the issuer's public key.

A NOTE ON TERMINOLOGY Damgård et al. call the problem at hand "unclonable group identification," meaning that, should a user make a copy of his sensor, the existence of such a clone will manifest itself when both sensors try to submit a piece of data in the same time period. We extend the problem, and call the extended version "periodic n-times anonymous authentication," because it is a technique that allows one to provide anonymous authentication up to n times during a given time period. For $n = 1$ (when there is only one e-token per user per time period) our scheme solves the same problem as the Damgård et al. scheme.

Related Work Anonymity, conditional anonymity, and revocable anonymity, are heavily researched fields; due to space constraints, we compare ourselves only to the most relevant and the most recent work. Anonymous credentials allow one to prove that one has a set of credentials without revealing anything other than this fact. Revocable anonymity [28, 11, 19, 38] allows a trusted third party to discover the identity of all otherwise anonymous participants; it is not directly relevant to our efforts since we do not assume any such TTP, nor do we want anyone to discover the identity of honest users. Conditional anonymity requires that a user's transactions remain anonymous until some conditions are violated; our results fall within that category. With the exception of Damgård et al.'s work [30], no prior literature on conditional anonymity considered conditions of the form "at most n anonymous transactions per time period are allowed." Most prior work on conditional anonymity focused on e-cash [27, 10, 13], where the identity of double-spenders could be discovered. A recent variation on the theme is Jarecki and Shmatikov's [37] work on anonymous, but *linkable*, authentication where one's identity can be discovered after one carries out too many transactions. Another set of recent papers [48, 49, 43, 3] addressed a related problem of allowing a user to show a credential anonymously and unlinkably up to k times to a particular verifier. In these schemes every verifier can set a different k. However, the k shows are always counted over the whole lifetime of the credential and not over limited time periods, as in our scheme.

2 Definition of Security

Our definitions for periodic n-times anonymous authentication are based on the e-cash definitions of [13] and [14]. We define a scheme where users $\mathcal{U}$ obtain e-token dispensers from the issuer $\mathcal{I}$, and each dispenser can dispense up to n anonymous and unlinkable e-tokens per time period, but no more; these e-tokens are then given to verifiers $\mathcal{V}$ that guard access to a resource that requires authentication (e.g., an on-line game). $\mathcal{U}$, $\mathcal{V}$, and $\mathcal{I}$ interact using the following algorithms:

– $\mathsf{IKeygen}(1^k, params)$ is the key generation algorithm of the e-token issuer $\mathcal{I}$. It takes as input 1^k and, if the scheme is in the common parameters model, these parameters $params$. It outputs a key pair $(pk_\mathcal{I}, sk_\mathcal{I})$. Assume that $params$ are appended as part of $pk_\mathcal{I}$ and $sk_\mathcal{I}$.
– $\mathsf{UKeygen}(1^k, pk_\mathcal{I})$ creates the user's key pair $(pk_\mathcal{U}, sk_\mathcal{U})$ analogously.
– $\mathsf{Obtain}(\mathcal{U}(pk_\mathcal{I}, sk_\mathcal{U}, n), \mathcal{I}(pk_\mathcal{U}, sk_\mathcal{I}, n))$ At the end of this protocol, the user obtains an e-token dispenser D, usable n times per time period and (optionally) the issuer obtains tracing information t_D and revocation information r_D. $\mathcal{I}$ adds t_D and r_D to a record $R_\mathcal{U}$ which is stored together with $pk_\mathcal{U}$.
– $\mathsf{Show}(\mathcal{U}(D, pk_\mathcal{I}, t, n), \mathcal{V}(pk_\mathcal{I}, t, n))$. Shows an e-token from dispenser D in time period t. The verifier outputs a token serial number (TSN) S and a transcript τ. The user's output is an updated e-token dispenser D'.
– $\mathsf{Identify}(pk_\mathcal{I}, S, \tau, \tau')$. Given two records (S, τ) and (S, τ') output by honest verifiers in the Show protocol, where $\tau \neq \tau'$, computes a value $s_\mathcal{U}$ that can identify the owner of the dispenser D that

generated TSN S.
The value $s_{\mathcal{U}}$ may also contain additional information specific to the owner of D that (a) will convince third parties that $\mathcal{U}$ is a violator (weak exculpability), that (b) will convince third parties that $\mathcal{U}$ double-showed this e-token (strong exculpability), or that (c) can be used to extract all token serial numbers of $\mathcal{U}$ (traceability).

A periodic n-times anonymous authentication scheme needs to fulfill the following three properties:

Soundness. Given an honest issuer, a set of honest verifiers are guaranteed that, collectively, they will not have to accept more than n e-tokens from a single e-token dispenser in a single time period. There is a knowledge extractor $\mathcal{E}$ that executes u Obtain protocols with all adversarial users and produces functions, $f_1, \ldots, f_u$, with $f_i : \mathbb{T} \times \mathbb{I} \rightarrow \mathbb{S}$. $\mathbb{I}$ is the index set $[0..n-1]$, $\mathbb{T}$ is the domain of the time period identifiers, and $\mathbb{S}$ is the domain of TSN's. Running though all $j \in \mathbb{I}$, $f_i(t, j)$ produces all n TSNs for dispenser i at time $t \in \mathbb{T}$. We require that for every adversary, the probability that an honest verifier will accept S as a TSN of a Show protocol executed in time period t, where $S \neq f_i(j, t)$, $\forall 1 \leq i \leq u$ and $\forall 0 \leq j < n$ is negligible.

Identification. There exists an efficient function ϕ with the following property. Suppose the issuer and verifiers $\mathcal{V}_1, \mathcal{V}_2$ are honest. If $\mathcal{V}_1$ outputs (S, τ) and $\mathcal{V}_2$ outputs (S, τ') as the result of Show protocols, then $\mathsf{Identify}(pk_{\mathcal{I}}, S, \tau, \tau')$ outputs a value $s_{\mathcal{U}}$, such that $\phi(s_{\mathcal{U}}) = pk_{\mathcal{U}}$, the violator's public key. In the sequel, when we say that a user has *reused* an e-token, we mean that there exist (S, τ) (S, τ') that are both output by honest verifiers.

Anonymity. An issuer, even when cooperating with verifiers and other dishonest users, cannot learn anything about an honest user's e-token usage behavior except what is available from side information from the environment. This property is captured by a simulator $\mathcal{S}$ which can interact with the adversary as if it were the user. $\mathcal{S}$ does not have access to the user's secret or public key, or her e-token dispenser D.

Formally, we create an adversary A that will play the part of the issuer and of all verifiers. A will create the public and private-keys of the issuer and verifiers. Then, A will be given access to an environment Env that is either using real users or a simulator; A must determine which. A can make four types of queries to Env:

$\mathsf{EnvSetup}(1^k)$ generates the public parameters $params$ (if any) and the private parameters $auxsim$ for the simulator (if there is one).

$\mathsf{EnvGetPK}(i)$ returns the public-key of user $\mathcal{U}_i$, generated by $\mathsf{UKeygen}(1^k, pk_{\mathcal{I}})$.

$\mathsf{EnvObtain}(i)$ runs the Obtain protocol with user $\mathcal{U}_i$: $\mathsf{Obtain}(\mathcal{U}(pk_{\mathcal{I}}, sk_{\mathcal{U}}, n), A(state))$. (We use state to denote whatever state the adversary maintains). We call D_j the dispenser generated the jth time protocol *Obtain* is run.

$\mathsf{EnvShow}(j, pk_{\mathcal{I}}, t)$ behaves differently depending on whether the environment is using a simulator. If the environment is using real users, it will simply run the Show protocol with the user $\mathcal{U}$ that holds the dispenser D_j: $\mathsf{Show}(\mathcal{U}(D_j, pk_{\mathcal{I}}, t, n), A(state))$. If the environment is using a simulator $\mathcal{S}$, then it will run the Show protocol with it: $\mathsf{Show}(\mathcal{S}(params, auxsim, pk_{\mathcal{I}}), A(state))$; $\mathcal{S}$ will not have access to the dispenser D_j or know who owns it.

An adversary is *legal* if it never asks a user to use the same dispenser to show more than n e-tokens in the same time-interval. We say that an e-token scheme preserves anonymity if no computationally bounded legal adversary can distinguish when the environment is playing with users and when it is using a simulator.

Additional Extensions In §6, we provide definitions of security in the context of *glitches*, i.e., re-use of e-tokens that do not occur "too often."

In §7, we discuss natural extensions to our basic construction that build on prior work on anonymous credentials and e-cash, namely the concepts of weak and strong exculpability, tracing, and revocation. We now define the corresponding algorithms and security guarantees for these extensions:

- $\mathsf{VerifyViolator}(pk_{\mathcal{I}}, pk_{\mathcal{U}}, s_{\mathcal{U}})$ publicly verifies that the user with public key $pk_{\mathcal{U}}$ has double-spent at least one e-token.
- $\mathsf{VerifyViolation}(pk_{\mathcal{I}}, S, pk_{\mathcal{U}}, s_{\mathcal{U}})$ publicly verifies that the user with public key $pk_{\mathcal{U}}$ is guilty of double-spending the e-token with TSN S.
- $\mathsf{Trace}(pk_{\mathcal{I}}, pk_{\mathcal{U}}, s_{\mathcal{U}}, R_{\mathcal{U}}, n)$, given a valid proof $s_{\mathcal{U}}$ and the user's tracing record $R_{\mathcal{U}}$, computes all TSNs corresponding to this user. Suppose the user has obtained u e-token dispensers, Trace outputs functions $f_1, \ldots, f_u$ such that by running though all $j \in [0..n-1]$, $f_i(t, j)$ produces all n TSNs for e-token dispenser D_i at time t. If $s_{\mathcal{U}}$ is invalid, i.e. $\mathsf{VerifyViolator}(pk_{\mathcal{I}}, pk_{\mathcal{U}}, s_{\mathcal{U}})$ rejects, Trace does nothing.
- $\mathsf{Revoke}(pk_{\mathcal{I}}, r_D, RD)$ takes as input a revocation database RD (initially empty) and revocation information r_D that corresponds to a particular user (see Obtain). It outputs the updated revocation database RD. In the sequel, we assume that RD is part of $pk_{\mathcal{I}}$.

These algorithms should fulfill the following properties:

Weak exculpability. An adversary cannot successfully blame an honest user $\mathcal{U}$ for reusing an e-token. More specifically, suppose an adversary can adaptively direct a user $\mathcal{U}$ to obtain any number of dispensers and show up to n e-tokens per dispenser per time period. The probability that the adversary produces $s_{\mathcal{U}}$ such that $\mathsf{VerifyViolator}(pk_{\mathcal{I}}, pk_{\mathcal{U}}, s_{\mathcal{U}})$ accepts is negligible.

Strong exculpability. An adversary cannot successfully blame a user $\mathcal{U}$ of reusing an e-token with token serial number S, even if $\mathcal{U}$ double-showed some other e-tokens. More specifically, suppose an adversary can adaptively direct a user to obtain any number of dispensers and show any number of e-tokens per dispenser per time period (i.e. he can reset the dispenser's state so that the dispenser reuses some of its e-tokens). The probability that the adversary outputs a token serial number S that was *not* reused and a proof $s_{\mathcal{U}}$ such that $\mathsf{VerifyViolation}(pk_{\mathcal{I}}, S, pk_{\mathcal{U}}, s_{\mathcal{U}})$ accepts is negligible.

Tracing of violators. The token serial numbers of violators can be efficiently computed. More specifically, given a value $s_{\mathcal{U}}$ such that $\mathsf{VerifyViolator}(pk_{\mathcal{I}}, pk_{\mathcal{U}}, s_{\mathcal{U}}, n)$ accepts, and supposing $\mathcal{U}$ has obtained u e-token dispensers, $\mathsf{Trace}(pk_{\mathcal{I}}, pk_{\mathcal{U}}, s_{\mathcal{U}}, R_{\mathcal{U}}, n)$ produces functions $f_1, \ldots, f_u$ such that by running though all $j \in [0..n-1]$, $f_i(t, j)$ produces all n TSNs for e-token dispenser i at time t.

Dynamic revocation. The Show protocol will only succeed for dispensers D that have not been revoked with Revoke. (Recall that Show takes as input the value $pk_{\mathcal{I}}$ that contains the database DB of revoked users.)

3 Preliminaries

Notation. We write $\mathbb{G} = \langle g \rangle$ to denote that g generates the group $\mathbb{G}$.

Bilinear Maps. Let $\mathsf{Bilinear\_Setup}$ be an algorithm that, on input the security parameter 1^k, outputs the parameters for a bilinear map as $\gamma = (q, g_1, h_1, \mathbb{G}_1,\ g_2,\ h_2,\ \mathbb{G}_2,\ \mathbb{G}_T, \mathbf{e})$. Each group $\mathbb{G}_1 = \langle g_1 \rangle = \langle h_1 \rangle$, $\mathbb{G}_2 = \langle g_2 \rangle = \langle h_2 \rangle$, and $\mathbb{G}_T$ are of prime order $q \in \Theta(2^k)$. The efficient mapping $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is both: (*Bilinear*) for all $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$, and $a, b \in \mathbb{Z}_q^2$, $\mathbf{e}(g_1^a, g_2^b) = \mathbf{e}(g_1, g_2)^{ab}$; and (*Non-degenerate*) if g_1 is a generator of $\mathbb{G}_1$ and g_2 is a generator of $\mathbb{G}_2$, then $\mathbf{e}(g_1, g_2)$ generates $\mathbb{G}_T$.

Complexity Assumptions. The security of our scheme relies on the following assumptions:

Strong RSA Assumption [5, 35]: Given an RSA modulus n and a random element $g \in \mathbb{Z}_n^*$, it is hard to compute $h \in \mathbb{Z}_n^*$ and integer $e > 1$ such that $h^e \equiv g \bmod n$. The modulus n is of a special form pq, where $p = 2p' + 1$ and $q = 2q' + 1$ are safe primes.

Additionally, our constructions require *one* of y-DDHI or SDDHI, depending on the size of the system parameters. Alternatively, we can substitute DDH for either of these assumptions, where the cost is an increase in our time and space complexity by a factor roughly the security parameter.

y-Decisional Diffie-Hellman Inversion (y-DDHI) [6, 33]: Suppose that $g \in \mathbb{G}$ is a random generator of order $q \in \Theta(2^k)$. Then, for all probabilistic polynomial time adversaries $\mathcal{A}$,

$$\begin{aligned}&\Pr[a \leftarrow \mathbb{Z}_q^*;\ x_0 = g^{1/a};\ x_1 \leftarrow \mathbb{G};\ b \leftarrow \{0,1\};\\ &\quad b' \leftarrow \mathcal{A}(g, g^a, g^{a^2}, \ldots, g^{a^y}, x_b) : b = b'] < 1/2 + 1/\text{poly}(k).\end{aligned}$$

Strong DDH Inversion (SDDHI): Suppose that $g \in \mathbb{G}$ is a random generator of order $q \in \Theta(2^k)$. Let $\mathcal{O}_a(\cdot)$ be an oracle that, on input $z \in \mathbb{Z}_q^*$, outputs $g^{1/(a+z)}$. Then, for all probabilistic polynomial time adversaries $\mathcal{A}^{(\cdot)}$ that do not query the oracle on x,

$$\begin{aligned}&\Pr[a \leftarrow \mathbb{Z}_q^*;\ (x, \alpha) \leftarrow \mathcal{A}^{\mathcal{O}_a}(g, g^a);\ y_0 = g^{1/(a+x)};\ y_1 \leftarrow \mathbb{G};\\ &\quad b \leftarrow \{0,1\};\ b' \leftarrow \mathcal{A}^{\mathcal{O}_a}(y_b, \alpha) : b = b'] < 1/2 + 1/\text{poly}(k).\end{aligned}$$

We show that the SDDHI assumption holds in generic groups in §4.

Additionally, our constructions require *one* of the following assumptions. XDH requires non-supersingular curves, whereas SF-DDH may reasonably be conjectured to hold in any bilinear group.

External Diffie-Hellman Assumption (XDH) [36, 45, 41, 7, 4]: Suppose $\mathsf{Bilinear\_Setup}(1^k)$ produces the parameters for a bilinear mapping $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. The XDH assumption states that the Decisional Diffie-Hellman (DDH) problem is hard in $\mathbb{G}_1$.

Sum-Free Decisional Diffie-Hellman Assumption (SF-DDH) [32]: Suppose that $g \in \mathbb{G}$ is a random generator of order $q \in \Theta(2^k)$. Let L be any polynomial function of k. Let $\mathcal{O}_{\vec{a}}(\cdot)$ be an oracle that, on input a subset $I \subseteq \{1, \ldots, L\}$, outputs the value $g_1^{\beta_I}$ where $\beta_I = \prod_{i \in I} a_i$ for some $\vec{a} = (a_1, \ldots, a_L) \in \mathbb{Z}_q^L$. Further, let R be a predicate such that $R(J, I_1, \ldots, I_t) = 1$ if and only if $J \subseteq \{1, \ldots, L\}$ is DDH-independent from the I_i's; that is, when $v(I_i)$ is the L-length vector with a one in position j if and only if $j \in I_i$ and zero otherwise, then there are no three sets I_a, I_b, I_c

such that $v(J) + v(I_a) = v(I_b) + v(I_c)$ (where addition is bitwise over the integers). Then, for all probabilistic polynomial time adversaries $\mathcal{A}^{(\cdot)}$,

$$\begin{aligned}\Pr[\vec{a} = (a_1, \ldots, a_L) \leftarrow \mathbb{Z}_q^L; (J, \alpha) \leftarrow \mathcal{A}^{\mathcal{O}_{\vec{a}}}(1^k); y_0 = g^{\prod_{i \in J} a_i};\\ y_1 \leftarrow \mathbb{G}; b \leftarrow \{0,1\}; b' \leftarrow \mathcal{A}^{\mathcal{O}_{\vec{a}}}(y_b, \alpha) \ : \ b = b' \wedge\\ R(J, Q) = 1] < 1/2 + 1/\mathrm{poly}(k),\end{aligned}$$

where Q is the set of queries that $\mathcal{A}$ made to $\mathcal{O}_{\vec{a}}(\cdot)$.

Key Building Blocks. We summarize the necessary information about our system components.

DY Pseudorandom Function (PRF). Let $\mathbb{G} = \langle g \rangle$ be a group of prime order $q \in \Theta(2^k)$. Let a be a random element of $\mathbb{Z}_q^*$. Dodis and Yampolskiy [33] showed that $f_{g,a}^{DY}(x) = g^{1/(a+x)}$ is a pseudorandom function, under the y-DDHI assumption, when either: (1) the inputs are drawn from the restricted domain $\{0,1\}^{O(\log k)}$ only, or (2) the adversary specifies a polynomial-sized set of inputs from $\mathbb{Z}_q^*$ *before* a function is selected from the PRF family (i.e., before the value a is selected). For our purposes, we require something stronger: that the DY construction work for inputs drawn arbitrarily and adaptively from $\mathbb{Z}_q^*$.

Theorem 3.1 *In the generic group model, the Dodis-Yampolskiy PRF is adaptively secure for inputs in* $\mathbb{Z}_q^*$.

The proof of Theorem 3.1 follows from the SDDHI assumption; see §4.

Pedersen and Fujisaki-Okamoto Commitments. Recall the Pedersen commitment scheme [44], in which the public parameters are a group $\mathbb{G}$ of prime order q, and generators $(g_0, \ldots, g_m)$. In order to commit to the values $(v_1, \ldots, v_m) \in {\mathbb{Z}_q}^m$, pick a random $r \in \mathbb{Z}_q$ and set $C = \mathrm{PedCom}(v_1, \ldots, v_m; r) = g_0^r \prod_{i=1}^m g_i^{v_i}$.

Fujisaki and Okamoto [35] showed how to expand this scheme to composite order groups.

CL Signatures. Camenisch and Lysyanskaya [17] came up with a secure signature scheme with two protocols: (1) An efficient protocol for a user to obtain a signature on the value in a Pedersen (or Fujisaki-Okamoto) commitment [44, 35] without the signer learning anything about the message. (2) An efficient proof of knowledge of a signature protocol. Security is based on the Strong RSA assumption. Using bilinear maps, we can use other signature schemes [18, 7] for shorter signatures.

Verifiable Encryption. For our purposes, in a verifiable encryption scheme, the encrypter/prover convinces a verifier that the plaintext of an encryption under a known public key is equivalent to the value hidden in a Pedersen commitment.

Camenisch and Damgård [12] developed a technique for turning any semantically-secure encryption scheme into a verifiable encryption scheme.

Bilinear El Gamal Encryption. We require a cryptosystem where g^x is sufficient for decryption and the public key is $\phi(g^x)$ for some function ϕ. One example is the bilinear El Gamal cryptosystem [8, 2], which is semantically secure under the DBDH assumption; that is, given (g, g^a, g^b, g^c, Q), it is difficult to decide if $Q = \mathbf{e}(g,g)^{abc}$. DBDH is implied by y-DDHI or Sum-Free DDH.

3.1 Agreeing on the Time.

Something as natural as time becomes a complex issue when it is part of a security system. First, it is necessary that the value of time period identifier t be the same for all users that show e-tokens in that period. Secondly, it should be used only for a single period, i.e., it must be unique. Our construction in §5 allows for the use of arbitrary time period identifiers, such as those negotiated using the hash tree protocol in [30]. For Glitch protection, §6, we assume a totally ordered set of time period identifiers.

If all parties have perfect clocks, then the current time (truncated in order to get the desired period size) fulfills all required properties. Since perfect clocks may be an unrealistic assumption, one of the parties must be motivated to enforce correctness. It is in the interest of verifiers to ensure that all users that show an e-token shown during a particular time period use the same time period identifier; otherwise, a dishonest user could create extra e-tokens within one time period by using different time period identifiers. Users are also interested in ensuring they all use the same time period identifier; otherwise, a verifier could link e-tokens that use similarly biased time period identifiers. In addition, users have a strong interest in ensuring that time period identifiers are unique, i.e. that they are never reused. Otherwise, e-tokens from different time periods would look like clones (even if they are not) and a verifier will be able to learn the user's identity.

Damgård et al. [30] describe a protocol that allows multiple users and a single verifier to agree on the same unique value for a time period identifier by having every user contribute sufficient randomness (i.e. uniqueness) to it. Their solution allows users to agree on a unique value without keeping additional state. Their approach can be used in our basic system §5, and for glitch protection §6.1, but not in window glitch protection §6.2, which requires that time period identifiers conform to a metric, i.e. that it is possible to compute the value for future time period identifiers from the current one.

Practice: In the real world, uniqueness can be enforced by checking that the system clock has not been turned back since the last invocation. Sameness for a given global period is more difficult to ensure. It is impossible to have a global notion of time in a distributed systems, so the only thing we can hope for, is to get the processes to agree on the time within a specific bound. Thus, this remains a possible line of attack for cheating verifiers. The situation can be improved by avoiding running the protocol at period boundaries.

Another practical decision is whether we want to have users announce the time, and verifiers check it, or whether we want to have verifiers announce the time and users check it. Each approach allows different attacks, e.g., by manipulating the users' time servers.

4 Generic Group Security of Full DY PRF

To provide more confidence in our scheme, we prove lower bounds on the complexity of our assumptions for generic groups [42, 46]. We follow the notation of Boneh and Boyen [6].

We will give the adversary as much power as possible and consider the case where we set SDDHI in $\mathbb{G}_1$ where $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ and the XDH assumption holds in $\mathbb{G}_1$. This means that there can be no efficiently computable homomorphism from $\mathbb{G}_1$ to $\mathbb{G}_2$. Our proof implies that the SDDHI assumption holds for non-bilinear generic groups as well.

In the generic group model, elements of the bilinear groups $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ are encoded as unique random strings. Thus, the adversary cannot directly test any property other than equality. Oracles are assumed to perform operations between group elements, such as performing the group operations in $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$. The opaque encoding of the elements of $\mathbb{G}_1$ is defined as the function $\xi_1 : \mathbb{Z}_p \to \{0,1\}^*$, which maps all $a \in \mathbb{Z}_p$ to the string representation $\xi_1(a)$ of $g_1^a \in \mathbb{G}_1$. Likewise, we have $\xi_2 : \mathbb{Z}_p \to \{0,1\}^*$ for $\mathbb{G}_2$ and $\xi_T : \mathbb{Z}_p \to \{0,1\}^*$ for $\mathbb{G}_T$. The adversary $\mathcal{A}$ communicates with the oracles using the ξ-representations of the group elements only.

Theorem 4.1 (SDDHI is Hard in Generic Groups) *Let $\mathcal{A}$ be an algorithm that solves the SDDHI problem in the generic group model, making a total of q_G queries to the oracles computing the group action in $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ of order p, the oracle computing the bilinear map $\mathbf{e}$, and oracle $\mathcal{O}_a(\cdot)$ that on input $i \in \mathbb{Z}_p^*$, outputs $g_1^{1/(a+i)}$. If $a \in \mathbb{Z}_p^*$ and ξ_1, ξ_2, ξ_T are chosen at random, then, when $\mathcal{A}$ does not query $\mathcal{O}$ on x,*

$$\Pr[(x,\alpha) \leftarrow \mathcal{A}^{\mathcal{O}_a}(p, \xi_1(1), \xi_1(a), \xi_2(1));\ y_0 = \xi_1(1/(a+x));\ r \leftarrow \mathbb{Z}_p^*;\ y_1 = \xi_1(r);\ b \leftarrow \{0,1\};$$
$$b' \leftarrow \mathcal{A}^{\mathcal{O}_a}(y_b, \alpha) : b = b'] \le \frac{1}{2} + 2 \cdot \frac{(q_G+3)^3}{p-1} = \frac{1}{2} + O\left(\frac{q_G^3}{p}\right).$$

Proof. Consider an algorithm $\mathcal{B}$ that interacts with $\mathcal{A}$ in the following game.

$\mathcal{B}$ maintains three lists of pairs $L_1 = \{(F_{1,i}, \xi_{1,i}) : i = 0, \ldots, \tau_1 - 1\}$, $L_2 = \{(F_{2,i}, \xi_{2,i}) : i = 0, \ldots, \tau_2 - 1\}$, and $L_T = \{(F_{T,i}, \xi_{T,i}) : i = 0, \ldots, \tau_T - 1\}$, such that, at step τ in the game, we have $\tau_1 + \tau_2 + \tau_T = \tau + 3$. Let the $F_{1,i}, F_{2,i}, F_{T,i}$ be *rational functions* (i.e, fractions whose numerators and denominators are polynomials); and all polynomials are multivariate polynomials in $\mathbb{Z}_p[a, r]$. The $\xi_{1,i}$, $\xi_{2,i}$, and $\xi_{T,i}$ are random strings in $\{0,1\}^*$ such that no two different polynomials are assigned the same string. We start the SDDHI game at step $\tau = 0$ with $\tau_1 = 2$, $\tau_2 = 1$ and $\tau_T = 0$. These correspond to the polynomials $F_{1,0} = 1$, $F_{1,1} = a$, $F_{2,0} = 1$ and the random strings $\xi_{1,0}, \xi_{1,1}, \xi_{2,0}$.

$\mathcal{B}$ begins the game with $\mathcal{A}$ by providing it with the 3 strings $\xi_{1,0}, \xi_{1,1}, \xi_{2,0}$. Now, we describe the oracles $\mathcal{A}$ may query.

Group action: $\mathcal{A}$ inputs two group elements $\xi_{1,i}$ and $\xi_{1,j}$, where $0 \le i, j < \tau_1$, and a request to multiply/divide. $\mathcal{B}$ sets $F_{1,\tau_1} \leftarrow F_{1,i} \pm F_{1,j}$. If $F_{1,\tau_1} = F_{1,u}$ for some $u \in \{0, \ldots, \tau_1 - 1\}$, then $\mathcal{B}$ sets $\xi_{1,\tau_1} = \xi_{1,u}$; otherwise, it sets ξ_{1,τ_1} to a unique random string in $\{0,1\}^* \setminus \{\xi_{1,0}, \ldots, \xi_{1,\tau_1-1}\}$. Finally, $\mathcal{B}$ returns ξ_{1,τ_1} to $\mathcal{A}$, adds $(F_{1,\tau_1}, \xi_{1,\tau_1})$ to L_1, and increments τ_1. Group actions for $\mathbb{G}_2$ and $\mathbb{G}_T$ are handled the same way.

Bilinear Map (e): $\mathcal{A}$ inputs two group elements $\xi_{1,i}$ and $\xi_{2,j}$, where $0 \le i < \tau_1$ and $0 \le j < \tau_2$. $\mathcal{B}$ sets $F_{T,\tau_T} \leftarrow F_{1,i} \cdot F_{2,j}$. If $F_{T,\tau_T} = F_{T,u}$ for some $u \in \{0, \ldots, \tau_T - 1\}$, then $\mathcal{B}$ sets $\xi_{T,\tau_T} = \xi_{T,u}$; otherwise, it sets ξ_{T,τ_T} to a random string in $\{0,1\}^* \setminus \{\xi_{T,0}, \ldots, \xi_{T,\tau_T-1}\}$. Finally, $\mathcal{B}$ returns ξ_{T,τ_T} to $\mathcal{A}$, adds $(F_{T,\tau_T}, \xi_{T,\tau_T})$ to L_T, and increments τ_T.

Oracle $\mathcal{O}_a(\cdot)$: $\mathcal{A}$ inputs c in $\mathbb{Z}_p^*$, followed by $\mathcal{B}$ setting $F_{1,\tau_1} = (1/a + c)$. If $F_{1,\tau_1} = F_{1,u}$ for some $u \in \{0, \ldots, \tau_1 - 1\}$, then $\mathcal{B}$ sets $\xi_{1,\tau_1} = \xi_{1,u}$; otherwise, it sets ξ_{1,τ_1} to a random string in $\{0,1\}^* \setminus \{\xi_{1,0}, \ldots, \xi_{1,\tau_1-1}\}$. $\mathcal{B}$ sends ξ_{1,τ_1} to $\mathcal{A}$, adding $(F_{1,\tau_1}, \xi_{1,\tau_1})$ to L_1. Finally, $\mathcal{B}$ adds one to τ_1.

Eventually $\mathcal{A}$ pauses and outputs a value $x \in \mathbb{Z}_p^*$ and an arbitrary state string α. $\mathcal{B}$ sets $F_{1,\tau_1} = r$, for a new variable r, sets ξ_{1,τ_1} to a random string in $\{0,1\}^* \setminus \{\xi_{1,0}, \ldots, \xi_{1,\tau_1-1}\}$, updates L_1, returns $(\xi_{1,\tau}, \alpha)$ to $\mathcal{A}$, and increments τ_1.

Next, $\mathcal{B}$ allows $\mathcal{A}$ to continue querying the above oracles. (Recall that at no time was $\mathcal{A}$ allowed to query $\mathcal{O}_a$ on x.) Eventually, $\mathcal{A}$ stops and outputs a bit b'. This is $\mathcal{A}$'s guess as to whether r is $1/(a+x)$ or an independent variable, although $\mathcal{B}$ has not yet decided this.

Notice that $\mathcal{A}$ cannot use the bilinear map to test his challenge value, because both a and r are in $\mathbb{G}_1$. Furthermore, notice that $\mathcal{A}$ cannot compute a representation in $\mathbb{G}_1$ corresponding to the polynomial $1/(a+x)$, for a direct comparison, unless he queries the oracle for it, which is forbidden. To see this, consider that using all the available oracles $\mathcal{A}$ can only compute polynomials $F_{1,i}(a,r) = c_0 + c_1 \cdot a + c_2 \cdot r + \sum_{j=1}^{n} \frac{d_j}{a+x_j}$, where the c_0, c_1, c_2, and d_j are integer constants chosen by $\mathcal{A}$ and the x_j are values with which it queries oracle $\mathcal{O}_a(\cdot)$. In particular $\mathcal{A}$ cannot compute $F_{1,i}(a,r) = \frac{1}{a+x_0}$ for some x_0, such that x_0 differs from all queried x_j. To see this we look at the polynomial equation

$$c_0 + c_1 \cdot a + c_2 \cdot r + \sum_{j=1}^{n} \frac{d_j}{a+x_j} - \frac{1}{a+x_0} = 0.$$

By multiplying out the denominators, we have

$$c_0 \cdot \prod_{j=0}^{n}(a+x_j) + c_1 \cdot a \cdot \prod_{j=0}^{n}(a+x_j) + c_2 \cdot r \cdot \prod_{j=0}^{n}(a+x_j) + \sum_{k=1}^{n} d_k \cdot \left(\prod_{\substack{j=1 \\ j \neq k}}^{n} (a+x_j) \right) \cdot (a+x_0) - \prod_{j=1}^{n}(a+x_j) = 0.$$

As we are operating only over polynomials in a and r, we are not yet concerned with divisions by zero or random solutions to the numeric equation. Looking only at the polynomial, we can see that c_0, c_1, and c_2 must be zero, as otherwise the terms for a^{n+2},a^{n+1}, and $r \cdot a^{n+1}$ cannot cancel out.

As a result we can simplify the equation to

$$(a+x_0) \cdot \sum_{k=1}^{n} d_k \cdot \left(\prod_{\substack{j=1 \\ j \neq k}}^{n} (a+x_j) \right) = \prod_{j=1}^{n}(a+x_j).$$

In order for the left and the right side to be equal, the corresponding polynomials must have the same linear factors. Note that $(a+x_0)$ is a factor on the left side and that the factors on the right correspond to the oracle queries. However, this means that x_0 must be equal to x_j for some $1 \leq j \leq n$. Thus, we have a contradiction, for $x_1, \ldots, x_n$ are the values with which $\mathcal{A}$ queried the oracle.

A similar argument holds for polynomials $F_{T,i}(a,r)$. First, consider that as $\mathbb{G}_2$ starts only with $F_{2,0} = 1$ and allows only for group operations all $F_{2,i}$ are of the form $F_{2,i}(a,r) = c$, for some constant c. Consequently the pairing oracle operating on two polynomials $F_{1,i}$, and $F_{2,i'}$ will only result in a polynomial that is again of the form $F_{1,\tau_T} = c_0 + c_1 \cdot a + c_2 \cdot r + \sum_{j=1}^{n} \frac{d_j}{a+x_j}$. Similarly, an optional homomorphism from $\mathbb{G}_2$ to $\mathbb{G}_1$ does not influence the type of polynomials an adversary can compute.

We have shown that $\mathcal{A}$ cannot compute a polynomial $F_{1,i}(a) = 1/(x_0 + a)$ or $F_{T,i}(a,r) = 1/(x_0+a)$. Consequently $\mathcal{B}$ can flips a coin b to set r arbitrarily, either to $1/(x_0+a)$ or to a random value. $\mathcal{A}$ has exactly $1/2$ chance of solving SDDHI, provided that $\mathcal{B}$'s simulation of this game works perfectly for any setting of a, r and b. We now evaluate this simulation.

Analysis of $\mathcal{B}$'s Simulation. In the simulation $\mathcal{B}$ chooses a random $a^* \in \mathbb{Z}_p^*$, $r^* \in \mathbb{Z}_p^*$, and b only after the $\mathcal{A}$ output his bit b'. $\mathcal{B}$ sets $a = a^*$ and sets $r = 1/(a^*+x)$ or $r = r^*$ depending on the value of b. $\mathcal{B}$ now tests (in equations (1), (2) for $b=0$ and (3), (4) for $b=1$) if its simulation was perfect; that is, if the instantiation of a, r, and b does *not* create any equality relation among the polynomials that was not revealed by the random strings provided to $\mathcal{A}$. Thus, $\mathcal{A}$'s overall success is bounded by the probability that any of the following holds:

$$F_{1,i}(a^*,r^*) - F_{1,j}(a^*,r^*) = 0, \quad \text{for some } i,j \text{ such that } F_{1,i} \neq F_{1,j}, \tag{1}$$
$$F_{T,i}(a^*,r^*) - F_{T,j}(a^*,r^*) = 0, \quad \text{for some } i,j \text{ such that } F_{T,i} \neq F_{T,j}, \tag{2}$$
$$F_{1,i}(a^*,\frac{1}{a^*+x}) - F_{1,j}(a^*,\frac{1}{a^*+x}) = 0, \quad \text{for some } i,j \text{ such that } F_{1,i} \neq F_{1,j}, \tag{3}$$
$$F_{T,i}(a^*,\frac{1}{a^*+x}) - F_{T,j}(a^*,\frac{1}{a^*+x}) = 0, \quad \text{for some } i,j \text{ such that } F_{T,i} \neq F_{T,j}, \tag{4}$$

The polynomials $F_{2,i}$ do not need to be considered, as they are constant and do not depend on a or r. We consider the case that $(a^* - x_i) = 0$, for some $0 \leq i \leq n$. This however happens only with probability at most $\tau_1/(p-1)$ and corresponds to randomly guessing a^*. In this case the simulation fails. Otherwise we look at the degree of the resulting polynomials when these rational fractions are summed and the denominators are multiplied out. Equation (1), (2) and (3), (4) have degree at most $n+1 \leq \tau_1$ and $n+2 \leq \tau_1$ respectively. For fixed i and j, we thus satisfy equations (1), (2), (3), and (4) each with probability $\leq \tau_1/(p-1)$.

Now summing over all (i,j) pairs in each case, we bound $\mathcal{A}$'s overall success probability $\varepsilon \leq \binom{\tau_1}{2}\frac{\tau_1}{p-1} + \binom{\tau_T}{2}\frac{\tau_1}{p-1} = \frac{\tau_1}{p-1} \cdot (\binom{\tau_1}{2} + \binom{\tau_T}{2})$. After adding the probability of a division by zero, the error probability of the simulation is less than $\frac{\tau_1}{p-1} + (1 - \frac{\tau_1}{p-1}) \cdot \frac{\tau_1}{p-1} \cdot (\binom{\tau_1}{2} + \binom{\tau_T}{2})$. Since $\tau_1 < p-1$ and $\tau_1 + \tau_2 + \tau_T \leq q_G + 3$, we simplify as follows:

$$\begin{aligned}
\varepsilon &\leq \frac{q_G+3}{p-1} + \frac{q_G+3}{p-1} \cdot \left(\frac{(q_G+3)\tau_1}{2} + \frac{(q_G+3)\tau_T}{2} \right) \\
&\leq \frac{q_G+3}{p-1} + \frac{q_G+3}{p-1} \cdot \frac{(q_G+3)(\tau_1+\tau_T)}{2} \\
&\leq (q_G+3)/(p-1) + (q_G+3)^3/(p-1) \\
&\leq 2 \cdot (q_G+3)^3/(p-1) = O(q_G^3/p).
\end{aligned}$$

The following corollary is immediate. □

Corollary 4.2 *Any adversary that breaks the SDDHI assumption with constant probability* $1/2 + \varepsilon > 0$ *in generic groups of order* p *requires* $\Omega(\sqrt[3]{\varepsilon p})$ *generic group operations.*

5 A Periodic n-Times Anonymous Authentication Scheme

5.1 Intuition Behind our Construction

In a nutshell, the issuer and the user both have key pairs. Let the user's keypair be $(pk_{\mathcal{U}}, sk_{\mathcal{U}})$, where $pk_{\mathcal{U}} = g^{sk_{\mathcal{U}}}$ and g is a generator of some group G of known order. Let f_s be a pseudorandom function whose range is the group G. During the Obtain protocol, the user obtains an e-token dispenser D that allows her to show up to n tokens per time period. The dispenser D is comprised of seed s for PRF f_s, the user's secret key $sk_{\mathcal{U}}$, and the issuer's signature on $(s, sk_{\mathcal{U}})$. We use CL signatures to prevent the issuer from learning anything about s or $sk_{\mathcal{U}}$. In the Show protocol, the user shows her i^{th} token in time period t: she releases TSN $S = f_s(0, t, i)$, a double-show tag $E = pk_{\mathcal{U}} \cdot f_s(1, t, i)^R$ (for a random R supplied by the verifier), and runs a ZK proof protocol that (S, E) correspond to a valid dispenser for time period t and $0 \leq i < n$ (the user proves that S and E were properly formed from values $(s, sk_{\mathcal{U}})$ signed by the issuer). Since f_s is a PRF, and all the proof protocols are zero-knowledge, it is computationally infeasible to link the resulting e-token to the user, the dispenser D, or any other e-tokens corresponding to D. If a user shows $n + 1$ e-tokens during the same time interval, then two of the e-tokens *must* use the same TSN. The issuer can easily detect the violation and compute $pk_{\mathcal{U}}$ from the two double-show tags, $E = pk_{\mathcal{U}} \cdot f_s(1, t, i)^R$ and $E' = pk_{\mathcal{U}} \cdot f_s(1, t, i)^{R'}$. From the equations above, $f_s(1, t, i) = (E/E')^{(R-R')^{-1}}$ and $pk_{\mathcal{U}} = E/f_s(1, t, i)^R$.

5.2 Our Basic Construction

Let k be a security parameter and $l_q \in O(k)$, l_x, l_{time}, and l_{cnt} be system parameters such that $l_q \geq l_x \geq l_{\text{time}} + l_{\text{cnt}} + 2$ and $2^{l_{\text{cnt}}} - 1 > n$, where n is the number of tokens we allow per time period.

In the following, we assume implicit conversion between binary strings and integers, e.g., between $\{0,1\}^l$ and $[0, 2^l - 1]$. Let $F_{(g,s)}(x) := f^{DY}_{g,s}(x) := g^{1/(s+x)}$ for $x, s \in \mathbb{Z}_q{}^*$ and $\langle g \rangle = \mathbb{G}$ being of prime order q. For suitably defined l_{time}, l_{cnt}, and l_x define the function $c : \{0,1\}^{l_x - l_{\text{time}} - l_{\text{cnt}}} \times \{0,1\}^{l_{\text{time}}} \times \{0,1\}^{l_{\text{cnt}}} \to \{0,1\}^{l_x}$ as:

$$c(u, v, z) := \left(u 2^{l_{\text{time}}} + v\right) 2^{l_{\text{cnt}}} + z \ .$$

Issuer Key Generation: In $\mathsf{IKeygen}(1^k, params)$, the issuer $\mathcal{I}$ generates two cyclic groups:

1. A group $\langle \mathbf{g} \rangle = \langle \mathbf{h} \rangle = \mathbf{G}$ of composite order $\mathbf{p}'\mathbf{q}'$ that can be realized by the multiplicative group of quadratic residue modulo a special RSA modulus $N = (2\mathbf{p}'+1)(2\mathbf{q}'+1)$. In addition to CL signatures, this group will be needed for zero-knowledge proofs of knowledge used in the sequel. Note that soundness of these proof systems is computational only and assumes that the prover does not know the order of the group.
2. A group $\langle g \rangle = \langle \tilde{g} \rangle = \langle h \rangle = \mathbb{G}$ of prime order q with $2^{l_q - 1} < q < 2^{l_q}$.

The issuer must also prove in zero-knowledge that N is a special RSA modulus, and $\langle \mathbf{g} \rangle = \langle \mathbf{h} \rangle$ are quadratic residues modulo N. In the random oracle model, one non-interactive proof may be provided. In the plain model, the issuer must agree to interactively prove this to anyone upon request.

Furthermore, the issuer generates a CL signature key pair (pk, sk) set in group $\mathbf{G}$. The issuer's public-key will contain $(\mathbf{g}, \mathbf{h}, \mathbf{G}, g, \tilde{g},\ h, \mathbb{G}, pk)$, while the secret-key will contain all of the information.

User Key Generation: In $\mathsf{UKeygen}(1^k, pk_{\mathcal{I}})$, the user chooses a random $sk_{\mathcal{U}} \in \mathbb{Z}_q$ and sets $pk_{\mathcal{U}} = g^{sk_{\mathcal{U}}} \in \mathbb{G}$.

Get e-Token Dispenser: $\mathsf{Obtain}(\mathcal{U}(pk_{\mathcal{I}}, sk_{\mathcal{U}}, n),\ \mathcal{I}(pk_{\mathcal{U}},\ sk_{\mathcal{I}}, n))$. Assume that $\mathcal{U}$ and $\mathcal{I}$ have mutually authenticated. A user $\mathcal{U}$ obtains an e-token dispenser from an issuer $\mathcal{I}$ as follows:

1. $\mathcal{U}$ and $\mathcal{I}$ agree on a commitment C to a random value $s \in \mathbb{Z}_q$ as follows:
   (a) $\mathcal{U}$ selects s' at random from $\mathbb{Z}_q$ and computes $C' = \mathrm{PedCom}(sk_{\mathcal{U}}, s'; r) = g^{sk_{\mathcal{U}}} \tilde{g}^{s'} h^r$.
   (b) $\mathcal{U}$ sends C' to $\mathcal{I}$ and proves that it is constructed correctly.
   (c) $\mathcal{I}$ sends a random r' from $\mathbb{Z}_q$ back to $\mathcal{U}$.
   (d) Both $\mathcal{U}$ and $\mathcal{I}$ compute $C = C'\tilde{g}^{r'} = \mathrm{PedCom}(sk_{\mathcal{U}},\ s' + r'; r)$. $\mathcal{U}$ computes $s = s' + r' \bmod q$.
2. $\mathcal{I}$ and $\mathcal{U}$ execute the CL signing protocol on commitment C. Upon success, $\mathcal{U}$ obtains σ, the issuer's signature on $(sk_{\mathcal{U}}, s)$. This step can be efficiently realized using the CL protocols [17, 18] in such a way that $\mathcal{I}$ learns nothing about $sk_{\mathcal{U}}$ or s.
3. $\mathcal{U}$ initializes counters $T := 1$ (to track the current period) and $J := 0$ (to count the e-tokens shown in the current time period). $\mathcal{U}$ stores the e-token dispenser $D = (sk_{\mathcal{U}}, s, \sigma, T, J)$.

Use an e-Token: $\mathsf{Show}(\mathcal{U}(E, pk_{\mathcal{I}}, t, n), \mathcal{V}(pk_{\mathcal{I}}, t, n))$. Let t be the current time period identifier with $0 < t < 2^{l_{\text{time}}}$. (We discuss how two parties might agree on t in Section 3.1.) A user $\mathcal{U}$ reveals a single e-token from a dispenser $D = (sk_{\mathcal{U}}, s, \sigma, T, J)$ to a verifier $\mathcal{V}$ as follows:

1. $\mathcal{U}$ compares t with T. If $t \neq T$, then $\mathcal{U}$ sets $T := t$ and $J := 0$. If $J \geq n$, abort!
2. $\mathcal{V}$ sends to $\mathcal{U}$ a random $R \in \mathbb{Z}_q^*$.
3. $\mathcal{U}$ sends to $\mathcal{V}$ a token serial number S and a double spending tag E computed as follows:

$$S = F_{(g,s)}(c(0, T, J)), \qquad E = pk_{\mathcal{U}} \cdot F_{(g,s)}(c(1, T, J))^R$$

4. $\mathcal{U}$ and $\mathcal{V}$ engage in a zero-knowledge proof of knowledge of values $sk_{\mathcal{U}}$, s, σ, and J such that:
   (a) $0 \leq J < n$,
   (b) $S = F_{(g,s)}(c(0, t, J))$,
   (c) $E = g^{sk_{\mathcal{U}}} \cdot F_{(g,s)}(c(1, t, J))^R$,
   (d) $\mathrm{VerifySig}(pk_{\mathcal{I}}, (sk_{\mathcal{U}}, s), \sigma)$=true.
5. If the proof verifies, $\mathcal{V}$ stores (S, τ), with $\tau = (E, R)$, in his database. If he is not the only verifier, he also submits this tuple to the database of previously shown e-tokens.
6. $\mathcal{U}$ increases counter J by one. If $J \geq n$, the dispenser is empty. It will be refilled in the next time period.

Technical Details. The proof in Step 4 is done as follows:

1. $\mathcal{U}$ generates the commitments $C_J = g^J h^{r_1}$, $C_u = g^{sk_{\mathcal{U}}} h^{r_2}$, $C_s = g^s h^{r_3}$, and sends them to $\mathcal{V}$.
2. $\mathcal{U}$ proves that C_J is a commitment to a value in the interval $[0, n-1]$ using standard techniques [23, 20, 9].
3. $\mathcal{U}$ proves knowledge of a CL signature from $\mathcal{I}$ for the values committed to by C_u and C_s in that order. This step can be efficiently realized using the CL protocols [17, 18].

4. $\mathcal{U}$ as prover and $\mathcal{V}$ as verifier engage in the following proof of knowledge, using the notation by Camenisch and Stadler [22]:

$$\begin{aligned} PK\{(\alpha, \beta, \delta, \gamma_1, \gamma_2, \gamma_3) : g = (C_s g^{c(0,t,0)} C_J)^{\alpha} h^{\gamma_1} \wedge \\ S = g^{\alpha} \ \wedge \ g = (C_s g^{c(1,t,0)} C_J)^{\beta} h^{\gamma_2} \wedge \\ C_u = g^{\delta} h^{\gamma_3} \wedge E = g^{\delta} (g^R)^{\beta}\} \ . \end{aligned}$$

$\mathcal{U}$ proves she knows the values of the Greek letters; all other values are known to both parties.

Let us explain the last proof protocol. From the first step we know that C_J encodes some value $\hat{J}$ with $0 \leq \hat{J} < n$, i.e., $C_J = g^{\hat{J}} h^{\hat{r}_J}$ for some $\hat{r}_J$. From the second step we know that C_s and C_u encoded some value $\hat{u}$ and $\hat{s}$ on which the prover $\mathcal{U}$ knows a CL signature by the issuer. Therefore, $C_s = g^{\hat{s}} h^{\hat{r}_s}$ and $C_u = g^{\hat{u}} h^{\hat{r}_u}$ for some $\hat{r}_s$ and $\hat{r}_u$. Next, recall that by definition of $c(\cdot,\cdot,\cdot)$ the term $g^{c(0,t,0)}$ corresponds to $g^{t2^{l_{\text{cnt}}}}$. Now consider the first term $g = (C_s g^{c(0,t,0)} C_J)^{\alpha} h^{\gamma_1}$ in the proof protocol. We can now conclude the prover $\mathcal{U}$ knows values $\hat{a}$ and $\hat{r}$ such that $g = g^{(\hat{s}+t2^{l_{\text{cnt}}}+\hat{J})\hat{a}} h^{\hat{r}}$ and $S = g^{\hat{a}}$. From the first equation is follows that $\hat{a} = (\hat{s} + (t2^{l_{\text{cnt}}} + \hat{J}))^{-1} \pmod{q}$ must hold provided that $\mathcal{U}$ is not privy to $\log_g h$ (as we show via a reduction in the proof of security) and thus we have established that $S = F_{(g,\hat{s})}(c(0,t,\hat{J}))$ is a valid serial number for the time period t. Similarly one can derive that $E = g^{\hat{u}} \cdot F_{(g,\hat{s})}(c(1,t,\hat{J}))^R$, i.e., that E is a valid double-spending tag for time period t.

Identify Cheaters: $\mathsf{Identify}(pk_{\mathcal{I}}, S, (E,R), (E',R'))$. If the verifiers who accepted these tokens were honest, then $R \neq R'$ with high probability, and proof of validity ensures that $E = pk_{\mathcal{U}} \cdot f_s(1,T,J)^R$ and $E' = pk_{\mathcal{U}} \cdot f_s(1,T,J)^{R'}$. The violator's public key can now be computed by first solving for $f_s(1,T,J) = (E/E')^{(R-R')^{-1}}$ and then computing $pk_{\mathcal{U}} = E/f_s(1,T,J)^R$.

Theorem 5.1 *Protocols* IKeygen*,* UKeygen*,* Obtain*,* Show*, and* Identify *described above achieve* soundness*,* identification*, and* anonymity *properties in the plain model assuming* Strong RSA*, and* y-DDHI *if* $l_x \in O(\log k)$ *or* SDDHI *otherwise.*

Proof. Soundness. Informally, in our system, tokens are unforgeable, because each token serial number (TSN) is a deterministic function $F_{(g,s)}(c(0,t,J))$ of the seed s, the time period t, and $J \in [0, n-1]$. Thus, there are only n valid TSNs per time period, and since a user must provide a ZK proof of validity for the token, to show $n+1$ or more times requires that two shows use the same TSN by the pigeonhole principle.

More formally, we will describe a knowledge extractor $\mathcal{E}$ that, after executing u Obtain protocols with an adversary $\mathcal{A}$ acting on behalf of all malicious users, can output functions $f_1, \ldots, f_u$ that allow to compute all possible token serial numbers that $\mathcal{A}$ could output in any given time period t. Let n be the number of shows allowed per time period. Our extractor $\mathcal{E}$ operates as follows:

1. In step one of Obtain, $\mathcal{E}$ behaves as an honest issuer and agrees on a Pedersen commitment $C = g^{sk_i} \tilde{g}^s h^r \ = \text{PedCom}(sk_i, s; r)$ with $\mathcal{A}$, where sk_i is whatever secret key $\mathcal{A}$ choses to use and s is the PRF seed.

2. In step two, $\mathcal{E}$ must run the CL signing protocol with $\mathcal{A}$ to provide $\mathcal{A}$ with a signature on (sk_i, s). As part of the CL protocol, $\mathcal{A}$ is required to prove knowledge of (α, β, γ) such that $C = g^{\alpha} \tilde{g}^{\beta} h^{\gamma}$. There are a number of ways to guarantee that this proof of knowledge is *extractable*;

in this step, $\mathcal{E}$ employs one of the methods of CL to extract the secrets $(sk_{\mathcal{U}}, s)$ from $\mathcal{A}$. (Here we will enforce that Obtain protocols must be run sequentially, so that rewinding does not become a problem.)

3. $\mathcal{E}$ outputs the function f_i as the description of the DY PRF (or whatever PRF is alternatively used) together with the seed s.

Since $\mathcal{E}$ knows the value s used for every dispenser, it can calculate the token serial number $S := F_{(g,s)}(c(0, t, J))$. The CL signatures and its protocols are secure under the Strong RSA assumption.

Identification of Violators. Suppose (S, E, R) and (S, E', R') are the result of two Show protocols with an honest verifier(s). Since the verifier(s) was honest, it is the case that $R \neq R'$ with high probability since an honest verifier chooses $R \in \mathbb{Z}_q^*$ at random. Due to the soundness of the ZK proof of validity, it must be the case that $E = pk_{\mathcal{U}} \cdot F_{(g,s)}(c(1, t, J))^R$ and $E' = pk_{\mathcal{U}} \cdot F_{(g,s)}(c(1, t, J))^{R'}$ for the same values of s, t, J and $pk_{\mathcal{U}}$. Thus, the violator's public key can be computed as follows:

$$\left(\frac{E^{1/R}}{E'^{1/R'}}\right)^{\frac{RR'}{R'-R}} = \left(\frac{g^{sk_{\mathcal{U}}/R} \cdot F_{(g,s)}(c(1,t,J))}{g^{sk_{\mathcal{U}}/R'} \cdot F_{(g,s)}(c(1,t,J))}\right)^{\frac{RR'}{R'-R}} = \left(g^{sk_{\mathcal{U}}(1/R - 1/R')}\right)^{\frac{RR'}{R'-R}} = g^{sk_{\mathcal{U}}} = pk_{\mathcal{U}}.$$

To be explicit with respect to our definition of this property, the value $s_{\mathcal{U}} := pk_{\mathcal{U}}$ and the function ϕ is the identity.

Anonymity. Informally, the intuition for anonymity is that the issuer does not learn the PRF seed during Obtain. Then showing a token in Show consists of releasing (S, E), where S and E are functions of the PRF (indistinguishable from random) and a zero-knowledge (ZK) proof that the token is valid (reveals one bit). Thus, if a user is honest, nothing about her identity is revealed by two random-looking numbers and a ZK proof.

More formally, will describe a simulator $\mathcal{S}$ which an adversary $\mathcal{A}$ cannot distinguish from an honest user during the Show protocol. Recall that $\mathcal{A}$, playing the role of a coalition of adversarial issuer and verifiers, first runs the Obtain protocol u times with honest users, who then obtain a dispenser D. Let the number of allowed shows per time period be n.

Now, at some point, $\mathcal{A}$ outputs a value $j \in [1, u]$ and a time period t. $\mathcal{A}$ will now execute the Show protocol with either the honest user $\mathcal{U}$ that holds the token dispenser D_j at time t *or* with simulator $\mathcal{S}$, whose input is only the global parameters *params*, t, n, and the issuer's public key $pk_{\mathcal{I}}$. To impersonate an unknown user, $\mathcal{S}$ behaves as follows:

1. In steps one and two of the Show protocol, $\mathcal{S}$ does nothing.

2. In step three, $\mathcal{S}$ sends to $\mathcal{A}$ random values $(S, E) \in \mathbb{G}^2$.

3. In step four, $\mathcal{S}$ *simulates* a proof of knowledge of (z, s, J, σ) for the statements:

   (a) $0 \le J < n$,
   (b) $S = F_{(g,s)}(c(0, t, J))$,
   (c) $E = g^z \cdot F_{(g,s)}(c(1, t, J))^R$.
   (d) VerifySig$(pk_{\mathcal{I}}, (z, s), \sigma)$=true.

Proving this statement, in the honest setting, follows the standard discrete-logarithm-based Σ-protocol, as we detailed in Section 5. Thus, for this step, $\mathcal{S}$ can simulate this Σ-protocol in the two standard ways: (1) rewind the adversary (interactive proof) or (2) use its control over the random oracle (non-interactive proof). To prevent any rewinding difficulties, Show protocols should be executed sequentially.

This simulator's behavior is indistinguishable from a user with dispenser D_j. The ZK proof is standard. The random values (S, E) are indistinguishable from the user's real (S', E') due to the security of the DY PRF $F_{(\cdot)}$, which relies on y-DDHI (for small system parameters) and otherwise SDDHI. Specifically, SDDHI is required for whenever parameter l_x becomes superlogarithmic due to a technicality in the original proof of security for the DY PRF. □

5.3 Efficiency Discussion

To analyze the efficiency of our scheme, it is sufficient to consider the number of (multi-base) exponentiations the parties have to do in $\mathbb{G}$ and $\mathbf{G}$. In a decent implementation, a multi-base exponentiation takes about the same time as a single-base exponentiation, provided that the number of bases is small. For the analysis we assume that the Strong RSA based CL-signature scheme is used.

Obtain: both the user and issuer perform 3 exponentiations in $\mathbf{G}$. Show: the user performs 12 multi-base exponentiation in $\mathbb{G}$ and 23 multi-base exponentiations in $\mathbf{G}$, while the verifier performs 7 multi-base exponentiation in $\mathbb{G}$ and 13 multi-base exponentiations in $\mathbf{G}$. If n is odd, the user only needs to do 12 exponentiations in $\mathbf{G}$, while the verifier needs to do 7. To compare ourselves to the Damgård et al. [30] scheme, we set $n = 1$. In this case, Show requires that the user perform 12 multi-base exponentiation in $\mathbb{G}$ and 1 multi-base exponentiations in $\mathbf{G}$ and the verifier perform 7 multi-base exponentiation in $\mathbb{G}$ and 1 multi-base exponentiations in $\mathbf{G}$. Damgård et al. requires $57+68r$ exponentiations in $\mathbb{G}$, where r is the security parameter (i.e., 2^{-r} is the probability that the user can cheat). Depending on the application, r should be at least 20 or even 60. Thus, our scheme is an order of magnitude more efficient than Damgård et al.

6 Glitch Protection Extension

In our periodic n-times anonymous authentication scheme, a user who shows two tokens with the same TSN becomes identifiable. (Recall that only n unique TSN values are available to a user per time period.) A user might accidentally use the same TSN twice because of hardware breakdowns, clock desychronization, etc. We want to protect the anonymity of users who occasionally cause a *glitch* (repeat a TSN in two different tokens), while still identifying users who cause an excessive amount of glitches. A user might be permitted up to m glitches per *monitoring interval* (e.g., year). Any TSN repetition will be detected, but the user's anonymity will not be compromised until the $(m + 1)$st glitch. A token that causes a glitch is called a *clone*.

Suppose a user has u glitches in one monitoring interval. Our goal is to design a scheme where:

- if $u = 0$, all shows are anonymous and unlinkable;
- if $1 \leq u \leq m$, all shows remain anonymous, but a link-id L is revealed, making all clones linkable;
- if $u > m$, the user's public key is revealed.

One can think of link-id L as a pseudonym (per monitoring interval) that is hidden in each token released by the same user (much in the same way that the user's public key was hidden in each token released by a user in the basic scheme). If tokens (S, τ) and (S, τ') caused a glitch, then we call (S, τ, τ') a *glitch tuple*, where by definition $\tau \neq \tau'$. We introduce a new function $\mathsf{GetLinkId}$ that takes as input a glitch tuple and returns the link-id L. Once $m+1$ clones are linked to the same pseudoym L, there is enough information from these collective original and cloned transcripts to compute the public key of the user.

We continue to use identifier $t \in \mathbb{T}$ for (indivisible) time periods. Identifier $v \in \mathbb{V}$ refers to a monitoring interval. We give two glitch protection schemes: §6.1 considers disjoint monitoring intervals, while §6.2 works on overlapping monitoring intervals. For the first scheme, we assume the existence of an efficient function $M_{\mathbb{V}}$ that maps every time period t to its unique monitoring interval $v \in \mathbb{V}$.

6.1 Basic Glitch Protection

Our basic glitch protection scheme tolerates up to m clones per monitoring interval v; monitoring intervals are disjoint.

We informally define the protocols and security properties of a periodic authentication scheme with glitch protection:

- $\mathsf{ShowGP}(\mathcal{U}(D, pk_{\mathcal{I}}, t, n, m), \mathcal{V}(pk_{\mathcal{I}}, t, n, m))$. Shows an e-token from dispenser D in time period t and monitoring interval $v = M_{\mathbb{V}}(t)$. The verifier obtains a token serial number S and a transcript τ.
- $\mathsf{GetLinkId}(pk_{\mathcal{I}}, S, \tau, \tau')$. Given e-tokens (S, τ, τ'), where $\tau \neq \tau'$ by definition, computes a link-id value L.
- $\mathsf{IdentifyGP}(pk_{\mathcal{I}}, (S_1, \tau_1, \tau_1'), \ldots, (S_{m+1}, \tau_{m+1}, \tau_{m+\ell+1}'))$.
  Given $m+1$ glitch tuples where for each i, $\mathsf{GetLinkId}(S_i, \tau_i, \tau_i')$ produces the same link-id L, computes a value $s_{\mathcal{U}}$ that can be used to compute the public key of the owner of the dispenser D from which the TSNs came.

We need to extend the anonymity and identification properties to handle the fact that honest users might occasionally make clones. Users that never clone should have the same security guarantees as they would in a basic anonymous authentication scheme without glitch protection: the result of every show must be *anonymous* and *unlinkable*. However, the verifier should be able to link clones together, but still be unable to identifier a user that has less than m glitches per monitoring interval.

Defining the extended anonymity property is tricky because we have to let the adversary request clones. We achieve this result by adding a fifth command to our environment: $\mathsf{EnvClone}(\mathit{token})$. The adversary controls which user shows it which e-token, what dispenser the user is supposed to use, the value of the counter J in the dispenser, and the time-interval for which this e-token was created. Therefore, this command gives it complete control over how and when users make clones. A scheme offers glitch protection if the adversary cannot distinguish between real users and a simulator, as long as it does not ask for too many clones (where too many is determined by the type of glitch protection).

There is a slight problem with this definition. The simulator is supposed to return a clone that is *linked* to some set of other clones generated by the same dispenser. The simulator can easily link clones of the same e-token because they have the same serial number. However, by the definition of anonymity, the simulator does not know what dispenser generated the e-token, nor what were that dispenser's other e-tokens.

Fortunately, the environment *can* keep track of this information. Therefore, the environment will pass an extra argument *linkid* to the simulator whenever the adversary invokes $\mathsf{EnvClone}$ (we make the environment do this rather than the adversary to ensure consistency). The *linkid* will be a unique integer independent of the user's id, dispenser, counter, etc. This *linkid* will let the simulator link only those e-tokens that ought to be linked. If the adversary never asks for clones, the anonymity property is the same as for n-times anonymous authentication without glitch protection.

Note: It's easy to see that whenever the adversary asks for an e-token, via either the $\mathsf{EnvShow}$ or $\mathsf{EnvClone}$ command, the environment can always consult its transcript of past communication to calculate $(\mathcal{U}, j, J, t)$, where $\mathcal{U}$ is the identity of the user, D_j is the dispenser that generated the e-token, J is the counter value D_j used, and t is the time-interval. Thus, we will assume that this information is available to the environment without explicitly calculating it. The environment will store a table *clones* that will count the number of clones of a particular type; it will use this table to ensure the adversary does not try to create more clones than is allowed by the glitch protection scheme. The environment will also have a table *LinkIDs* that will store the *linkid*s it will pass to the simulator.

Defining identification is also a little tricky because we need a way to *link* clones that were generated by different values (j, J, t) and to identify users from these clones. We create a new helper function $\mathsf{GetLinkId}(S, \tau_1, \tau_2)$ that outputs a value L that is the same for all clones made by the a dispenser in the same time interval. In otherwords, if a single dispenser made e-tokens (S, τ_1), (S, τ_2), (S', τ_1') and (S', τ_2'), then $\mathsf{GetLinkId}(S, \tau_1, \tau_2) = \mathsf{GetLinkId}(S', \tau_1', \tau_2')$. Using $\mathsf{GetLinkId}$, we can require that the function $\mathsf{IdentifyGP}$ return the public-key of the violator when it gets as input a sufficiently long sequence of clones that are all linked (as defined by $\mathsf{GetLinkId}$). It is upto the designers of a glitch-protection scheme to instantiate $\mathsf{GetLinkId}$ and ensure that it does not clash with the anonymity and glitch protection property.

We formally define the GP Anonymity property of an n-times anonymous authentication scheme with basic glitch protection.

GP Anonymity. The adversary will interact with the environment. The environment will generate a table *clones* that will count how many clones a dispenser j made in time interval $v \in \mathbb{V}$ and a corresponding list *LinkIDs*, such that $LinkIDs(j, v)$ is unique for every pair (j, v). The first time the adversary invokes $\mathsf{EnvShow}(j, *, t)$ (it does not matter which verifier the adversary uses), the environment will set $clones(j, M_{\mathbb{V}}(t)) = 0$. Whenever the adversary invokes $\mathsf{EnvClone}(etoken)$, before fulfilling the request, the environment will check if $clones(j, v) \geq m$. If yes, the environment will output *error*. Otherwise the environment will fulfill the request and increment $clones(j, v)$; if it is using a simulator, the environment will give the simulator $LinkIDs(j, v)$ as input.

We say that a scheme offers GP Anonymity if, in this game, no computationally bounded adversary can tell if the environment is using real users or a simulator.

GP Identification. Suppose the issuer and verifiers are honest and they receive $m+1$ glitch tuples $\mathsf{Input} = (S_1, \tau_1, \tau_1'), \ldots, (S_{m+1}, \tau_{m+1}, \tau_{m+1}')$ with the same $L = \mathsf{GetLinkId}(pk_{\mathcal{I}}, S_i, \tau_i, \tau_i')$ for all $1 \leq i \leq m+1$. Then with high probability algorithm $\mathsf{IdentifyGP}(pk_{\mathcal{I}}, \mathsf{Input})$ outputs a value $s_{\mathcal{U}}$ for which there exists an efficient function ϕ such that $\phi(s_{\mathcal{U}}) = pk_{\mathcal{U}}$, identifying the violator.

Intuition behind construction. Recall that in our basic scheme, an e-token has three logical parts: a serial number $S = F_{(s,g)}(c(0,T,J))$, a tag $E = pk_{\mathcal{U}} \cdot F_{(s,g)}(c(1,T,J))^R$, and a proof of validity. If the user shows a token with TSN S again, then he must reveal $E' = pk_{\mathcal{U}} \cdot F_{(s,g)}(c(1,T,J))^{R'}$, where $R \neq R'$, and the verifier can solve for $pk_{\mathcal{U}}$ from (E, E', R, R').

Now, in our glitch protection scheme, an e-token has four logical parts: a serial number $S = F_{(s,g)}(c(0,T,J))$, a tag K that exposes the link-id L if a glitch occurs, a tag E that exposes $pk_{\mathcal{U}}$ if more than m glitches occur, and a proof of validity.

We instantiate $K = L \cdot F_{(g,s)}(c(2,T,J))^R$. Now a double-show reveals L just as it revealed $pk_{\mathcal{U}}$ in the original scheme. The link-id for monitoring interval v is $L = F_{(s,g)}(c(1,v,0))$.

Once the verifiers get $m+1$ clones with the same link-id L, they need to recover $pk_{\mathcal{U}}$. To allow this, the user includes tag $E = pk_U \cdot \prod_{i=1}^{m} F_{(g,s)}(c(3,v,i))^{\rho_i} \cdot F_{(s,g)}(c(4,T,J))^R$. (Here, it will be critical for anonymity that the user and the verifier *jointly* choose the random values $R, \rho_1, \ldots, \rho_m$.)

Now, suppose a user causes $m+1$ glitches involving ℓ distinct TSNs. Given $(E, R, \rho_1, \ldots, \rho_m)$ from each of these $(m+\ell+1)$ tokens, the public key of the user can be computed by repeatedly using the elimination technique that allowed the discovery of L from (K, K', R, R'). We have $(m+\ell+1)$ equations E and $(m+\ell+1)$ unknown *bases* including $pk_{\mathcal{U}}$ and the $F_{(s,g)}(.)$ values. Thus, solving for $pk_{\mathcal{U}}$ simply requires solving a system of linear equations.

Construction. ShowGP and IdentifyGP replace the corresponding Show and Identify algorithms of the basic construction in §5.

ShowGP$(\mathcal{U}(D, pk_{\mathcal{I}}, t, n, m), \mathcal{V}(pk_{\mathcal{I}}, t, n, m))$. Let $v = M_{\mathbb{V}}(t)$. A user $\mathcal{U}$ shows a single e-token from a dispenser $D = (sk_{\mathcal{U}}, s, \sigma, T, J)$ to a verifier $\mathcal{V}$ as follows:

1. $\mathcal{U}$ compares t with T. If $t > T$, then $\mathcal{U}$ sets $T := t$ and $J := 0$. If $J \geq n$, abort!
2. $\mathcal{V}$ and $\mathcal{U}$ jointly choose $R, \rho_1, \ldots, \rho_m$ uniformly at random from $\mathbb{Z}_q^*$ (see §6.3 for details).
3. $\mathcal{U}$ sends $\mathcal{V}$ an interval serial number S, a double spending tag K encoding the link-id L, and a special $(m+1)$-cloning tag E:

$$\begin{aligned} S &= F_{(g,s)}(c(0,T,J)), \\ K &= F_{(g,s)}(c(1,v,0)) \cdot F_{(g,s)}(c(2,T,J))^R, \\ E &= pk_{\mathcal{U}} \cdot F_{(g,s)}(c(3,v,1))^{\rho_1} \cdots \\ &\qquad F_{(g,s)}(c(3,v,m))^{\rho_m} \cdot F_{(g,s)}(c(4,T,J))^R \end{aligned}$$

4. $\mathcal{U}$ performs a zero-knowledge proof that the values above were correctly computed.
5. If the proof verifies, $\mathcal{V}$ stores (S, τ), where $\tau = (K, E, R, \rho_1, \ldots, \rho_m)$, in his database.
6. $\mathcal{U}$ increments counter J by one. If $J \geq n$ the dispenser is empty. It will be refilled in the next time period.

GetLinkId$(pk_{\mathcal{I}}, S, (K, E, R, \vec{\rho}), (K', E', R', \vec{\rho'}))$. Returns

$$L = \frac{K}{(K/K')^{(R-R')^{-1}R}}.$$

IdentifyGP$(pk_{\mathcal{I}}, (S_1, \tau_1, \tau_1'), \ldots, (S_{m+1}, \tau_{m+1}, \tau_{m+1}'))$. Let the $m+1$ glitch tuples include ℓ distinct TSN values. We extract the values $(E_i, R, \rho_1, \ldots, \rho_m)$ (or $(E_i', R', \rho_1', \ldots, \rho_m')$) from all $m+\ell+1$ unique transcripts. Now, we use the intuition provided above to solve for $pk_{\mathcal{U}}$.

Theorem 6.1 *The scheme described above is a secure periodic n-times anonymous authentication scheme with basic glitch protection. It fulfills the soundness, GP anonymity and GP identification properties.*

Proof. Soundness: The soundness proof is identical to the soundness proof for the basic n-times anonymous authentication scheme.

GP Anonymity: All we need to do to prove GP anonymity is to explain how the simulator will respond to EnvSetup, EnvShow, and EnvClone.

To ensure consistency, the simulator will store a table $Token$ of information about e-tokens it has previously shown. It will also have a table $Link$ indexed by link-ids provided by the environment. $Link(linkid)$ will be a random number; the simulator will ensure that when the adversary runs GetLinkId on a double-show with link-id $linkid$, the output will always be $Link(linkid)$.

During EnvShow, the simulator will run the Show protocol with the adversary. The simulator will perform steps 1, 2 and 3 as normal; at the end of which it will have random $(R, \rho_1, \ldots, \rho_m)$. In step 4 The simulator will randomly choose $(S, \hat{L}, E) \in \mathbb{G}^3$. In step five, $\mathcal{S}$ *simulates* a proof of knowledge of (z, s, J, σ) for the statements:

(a) $0 \leq J < n$,
(b) $S = F_{(g,s)}(c(0, T, J))$
(c) $\hat{L} = F_{(g,s)}(c(1, v, 0)) \cdot F_{(g,s)}(c(2, T, J))^R$
(d) $E = pk_{\mathcal{U}}{}^{\rho_0} \cdot F_{(g,s)}(c(3, v, 1))^{\rho_1} \cdots F_{(g,s)}(c(3, v, m))^{\rho_m} \cdot F_{(g,s)}(c(4, T, J))^R$
(e) $\text{VerifySig}(pk_{\mathcal{I}}, (z, s), \sigma)$=true.

Simulating such a proof is a standard operation; see the anonymity proof in §5.2 for details. Call the e-token resulting from this execution $etoken$. The simulator will store $Token(etoken) = (S, R, \hat{L}, F = \phi)$. If the token is ever cloned with some link-id $linkid$, the simulator will retroactively set F so that $\hat{L} = Link(linkid) \cdot F^R$.

When the adversary invokes EnvClone($etoken$), the environment will give the simulator a $linkid$. Then the adversary and the simulator will run through the Show protocol. The simulator will have to produce an e-token that is a clone of $etoken$ and that is linked to all other e-tokens with link-id $linkid$.

Once again, the simulator will perform steps 1, 2 and 3 as normal; at the end of which it will have random $(R', R'_1, \ldots, R'_m)$. In step 4, the simulator will perform the following operations: first, it will retrieve $Token(etoken) = (S, R, \hat{L}, F)$. If $F = \phi$, then the simulator will calculate $F = (\hat{L}/Link(linkid))^{1/R}$ and update the entry $Token(etoken)$ accordingly. Then, the simulator will create a new e-token: $S' = S$, $\hat{L}' = Link(linkid)F^{R'}$, and E' will be a random number. In step 5, the simulator will simulate a proof of knowledge of (z, s, J, σ) for the statements:

(a) $0 \leq J < n$,
(b) $S' = F_{(g,s)}(c(0, T, J))$
(c) $\hat{L}' = F_{(g,s)}(c(1, v, 0)) \cdot F_{(g,s)}(c(2, T, J))^{R'}$
(d) $E' = pk_{\mathcal{U}}{}^{R'_0} \cdot F_{(g,s)}(c(3, v, 1))^{R'_1} \cdots F_{(g,s)}(c(3, v, m))^{R'_m} \cdot F_{(g,s)}(c(4, T, J))^{R'}$
(e) $\text{VerifySig}(pk_{\mathcal{I}}, (z, s), \sigma)$=true.

It will do this in the same manner as for Show. After terminating, the simulator will store $Token(etoken') = (S, R', \hat{L}', F)$.

We sketch out why the output of the simulator is indistinguishable from that of real users: Until the adversary asks for a clone, an e-token is a completely random number. It is generated in the same way as in the original proof of anonymity; therefore the output of the simulator is indistinguishable from that of real users. When the adversary asks for a clone, the simulator retroactively sets the link-id to be $Link(linkid)$, where $linkid$ is provided by the environment. This ensures that the link-ids are consistent. The $(m+1)$-cloning tags E are all random numbers; this is fine because the adversary never gets $m+1$ clones and therefore they should provide no information.

GP Identification: By the soundness property, we know that the glitch-tuples given to the function $\mathsf{IdentifyGP}(pk_{\mathcal{I}}, (S_1, \tau_1, \tau'_1), \ldots, (S_{m+1}, \tau_{m+1}, \tau'_{m+1}))$ are correctly formed. The only remaining question is whether the $(m+1)$-clone tags E_i^b contain enough information to solve for $pk_{\mathcal{U}}$.

Since the issuer never learns the seed of the PRF used to calculate the $m+1$-clone tags E_i^b, as far as it is concerned, each tag E_i^b is the product of $m+2$ randomly chosen unknowns. One of the unknowns is $pk_{\mathcal{U}}$, m of the unknowns are the same for every tag from that monitoring interval, and one of the unknowns is unique to the cloned e-token. Therefore, if a user clones ℓ different e-tokens ($|\{S_1, \ldots, S_{m+1}\}| = \ell$), then there are $m+1+\ell$ different unknowns among all the E_i^b. How many distinct tags E_i^b are there? If there are ℓ different multi-shown e-tokens, then there are exactly $m+1+\ell$ different tags E_i^b: there are ℓ distinct E_i and $m+1$ distinct E'_i. With high probability, a randomly generated system of $m+1+\ell$ equations with $m+1+\ell$ unknowns will have a unique solution, in which case GP Identify will find it using Gaussian Elimination.

More specifically if we represent the $y = m+\ell+1$ equations as a $y \times y$ matrix, the success probability P_S corresponds to the probability that a random matrix of this size is invertible. The first vector in your matrix is arbitrary, except that it should not be 0. So there are $(q^y - 1)$ choices. The second vector should be linearly independent of the first; i.e., it should not lie in a 1-dimensional subspace. So there are $(q^y - q)$ choices. The third vector should not lie in a 2-dimensional subspace, so there are $(q^y - q^2)$ choices. Etc. So in general, there are $(q^y-1)\cdot(q^y-q)\cdots(q^y - q^{(y-1)})$ invertible $y \times y$ matrices. To get the probability one divides this number by the total number of matrices, that is $q^{(y^2)}$. It is easy to see that

$$
\begin{aligned}
P_S &= \frac{(q^y-1)\cdot(q^y-q)\cdots(q^y-q^{(y-1)})}{q^{(y^2)}} = \left(1-\frac{1}{q^y}\right)\cdot\left(1-\frac{1}{q^{y-1}}\right)\cdots\left(1-\frac{1}{q}\right) \\
&\geq \left(1-\frac{1}{q}\right)\cdot\left(1-\frac{1}{q}\right)\cdots\left(1-\frac{1}{q}\right) \\
&= \left(1-\frac{1}{q}\right)^y = \sum_{i=0}^{y}(-1)^{(i \bmod 2)}\cdot\binom{y}{i}\cdot(1/q)^{y-i} \\
&= 1 + \sum_{i=1}^{y}(-1)^{(i \bmod 2)}\cdot\binom{y}{i}\cdot(1/q)^{y-i}.
\end{aligned}
$$

As q is exponential in the security parameter k, P_S is bounded below by $1 - neg(k)$, with $neg(k)$ a negligible function in k. □

6.2 Window Glitch Protection

The basic glitch protection scheme prevents users from creating more than m clones in a single monitoring interval. If two neighboring time periods fall in different monitoring intervals, then a malicious user can create m clones in each of them. We want to catch users who make more than m clones within any W consecutive time periods.

We define an interval of consecutive time-periods to be a window. For convenience, we will consider each time period identifier t to be an integer, and time periods t and $t+1$ to be neighbors. Each time period is in W different windows of size W. If we let a time period define the *end* of a window, then time period t would be in windows $t, t+1, \ldots, t+W-1$.

(m, W)-Window glitch protection allows a user to clone at most m e-tokens during any window of W consecutive time periods. We describe the new protocols associated with a window glitch protection scheme:

- $\mathsf{ShowWGP}(\mathcal{U}(D, pk_{\mathcal{I}}, t, n, m, W), \mathcal{V}(pk_{\mathcal{I}}, t, n, m, W))$.
  Shows an e-token from dispenser D for time period t. The verifier obtains a serial number S and a transcript τ.
- $\mathsf{GetLinkIds}(pk_{\mathcal{I}}, S, \tau, \tau')$. Given two e-tokens (S, τ) and (S, τ'), outputs a *list* of W link-ids $L_1, \ldots, L_W$.
- $\mathsf{IdentifyWGP}(pk_{\mathcal{I}}, (S_1, \tau_1, \tau_1'), \ldots, (S_{m+1}, \tau_{m+1}, \tau_{m+1}'))$.
  Given $m+1$ glitch tuples where for each i, the same link-id L is in the list of link-ids produced by $\mathsf{GetLinkId}(S_i, \tau_i, \tau_i')$, computes a value $s_{\mathcal{U}}$ that can be used to compute the public key of the owner of the dispenser D from which the TSNs came.

We modify the *GP Anonymity* and *GP Identification* properties to apply to window glitch protection.

WGP Anonymity. The environment will keep a table *clones* that will count how many clones every user made during every window of length W, and a table *LinkIDs* with a random unique entry for each time interval $i \in \mathbb{V}$. Each time the adversary invokes $\mathsf{EnvClone}$, before fullfilling the request, the environment will increment the values at $clones(\mathcal{U}, t), \ldots, clones(\mathcal{U}, t+w-1)$. If any of those result in a value greater than m, the environment will output *error*. Otherwise, the environment will run the Show protocol; if it is using the simulator, the environment will give it $LinkIDs(t), \ldots, LinkIds(t+w-1)$ as input.

WGP Identification. Suppose the issuer and verifiers are honest. Should they receive a list of $m+1$ glitch tuples $\mathsf{Input} = (S_1, \tau_1, \tau_2'), \ldots, (S_{m+1}, \tau_{m+1}, \tau_{m+1}')$, such that $\exists L : \forall i : L \in \mathsf{GetLinkIds}(pk_{\mathcal{I}}, S_i, \tau_i, \tau_i')$, then with high probability $\mathsf{IdentifyWGP}(pk_{\mathcal{I}}, \mathsf{Input})$ outputs a value $s_{\mathcal{U}}$ for which there exists an efficient function ϕ such that $\phi(s_{\mathcal{U}}) = pk_{\mathcal{U}}$, identifying the violator.

Construction. Intuitively, we replicate our basic glitch solution W times for overlapping windows of W time periods.

$\mathsf{ShowWGP}(\mathcal{U}(D, pk_{\mathcal{I}}, t, n, m, W)$. We modify the ShowGP protocol as follows. In step 3, the user and verifier jointly choose random numbers $R_1, \ldots, R_W$ and $\rho_{1,1}, \ldots, \rho_{W,m}$. In step 4, the user calculates essentially the same values S, K, E, except that now she calculates separate K_i and E_i

tags for every window in which time period T falls:

$$\begin{aligned}
S &= F_{(s,g)}(c(0,T,J)) \\
K_i &= F_{(s,g)}(c(1,T+i,0)) \cdot F_{(s,g)}(c(2,T,J))^{R_i} \\
E_i &= pk_{\mathcal{U}} \cdot F_{(s,g)}(c(3,T+i,1))^{\rho_{i,1}} \cdots \\
&\qquad F_{(s,g)}(c(3,T+i,m)))^{\rho_{i,m}} \cdot F_{(s,g)}(c(4,T,J))^{R_i}
\end{aligned}$$

Finally, in step 5, the user proves to the verifier that the values $S, K_1, \ldots, K_W, E_1, \ldots, E_W$ are formed correctly. That, along with the random numbers generated in step 3, forms the transcript stored in steps 6. Step 7 is unchanged.

$\mathsf{GetLinkIds}(pk_{\mathcal{I}}, S, \tau, \tau')$. Returns the link-ids:

$$L_i = \frac{K_i}{(K_i/K_i')^{(R_i-R_i')^{-1}R_i}}, \quad 1 \le i \le m+1.$$

$\mathsf{IdentifyWGP}(pk_{\mathcal{I}}, (S_1, \tau_1, \tau_2'), \ldots, (S_{m+1}, \tau_{m+1}, \tau_{m+1}'))$. For all i, let $L \in \mathsf{GetLinkIds}(pk_{\mathcal{I}}, S_i, \tau_i, \tau_i')$, that is, let L be the link-id each glitch tuple has in common. Let these $m+1$ glitch tuples include ℓ distinct TSN values. We extract the values $(E_{i,j}, R_i, \rho_{i,1}, \ldots, \rho_{i,m})$ (or $(E_{i,j}', R_i', \rho_{i,1}', \ldots, \rho_{i,m}')$) from all $m+\ell+1$ unique transcripts, where j depends on where L falls in the list $\mathsf{GetLinkIds}(pk_{\mathcal{I}}, S_i, \tau_i, \tau_i')$. Now, we use the same techniques as before to solve for $pk_{\mathcal{U}}$.

Theorem 6.2 *The scheme described above is a secure periodic n-times anonymous authentication scheme with window glitch protection. It fulfills the soundness, WGP anonymity and WGP identification properties.*

Proof. Soundness: The soundness proof is identical to the soundness proof for the basic n-times anonymous authentication scheme.

WGP Anonymity: The WGP anonymity proof follows closely after the GP anonymity proof in §6.1. Essentially, the only difference between basic glitch protection e-tokens and window glitch protection e-tokens is that we now have $\hat{L}_1, \ldots, \hat{L}_W$ instead of just one $\hat{L}$ and $E_1, \ldots, E_W$ instead of just E. Therefore, step 4 in both showing and cloning will construct each $\hat{L}_i$ and E_i using the same techniques as for $\hat{L}$ and E, and the simulated proof in step 5 will reflected the more complicated construction. We give a sketch:

Once again, the simulator will store tables $Token$ and $Link$ (though entries in $Token$ will be longer because e-tokens contain more data). During $\mathsf{EnvShow}$, the simulator will run the Show protocol in essentially the same manner. The simulator will perform steps 1, 2 and 3 as normal; at the end of which it will have random $R_1, \ldots, R_W, \rho_{1,1}, \ldots, \rho_{W,m}$. In step 4 the simulator will randomly choose $(S, \hat{L}_1, \ldots, \hat{L}_W, E_1, \ldots, E_W) \in \mathbb{G}^{1+2W}$. In step five, the simulator will fake the proof of knowledge of (z, s, J, σ) showing the e-token is formed correctly, using the technique as §5.2. The simulator will. store $Token(etoken) = (S, R_1, \ldots, R_W, \rho_{1,1}, \ldots, \rho_{W,m}, \hat{L}_1, \ldots, \hat{L}_W, F_1 = \phi, \ldots, F_W = \phi)$. If the token is ever cloned with some link-ids $linkid_1, \ldots, linkid_W$, the simulator will retroactively set all the F_i so that $\hat{L}_i = Link(linkid_i) \cdot F_i^{R_i}$.

The simulator also clones e-tokens in the same way as the simulator for GP anonymity. It looks up the original e-token in $Token(etoken)$. In step 4, the simulator uses $linkid_i$ to calculate the value for F_i, and from that calculates the $\hat{L}_i$. Then it randomly chooses the E_i. In step 5 it once again simulates a zero-knowledge proof of knowledge showing the e-token is correctly formed. Finally, it stores the new e-token in $Token(etoken')$.

WGP Identification: Observe that $\mathsf{IdentifyWGP}$ is identical to $\mathsf{IdentifyGP}$, once the appropriate link-id L is chosen. If there are $\ell = |\{S_1, \ldots, S_{m+1}\}|$ different cloned e-tokens, then there are $m + 1 + \ell$ distinct tags E^b_{i,j_i} and $m + 1 + \ell$ unknowns. Therefore, with high probability, the system of equations can be solved via Gaussian Elimination to reveal $pk_{\mathcal{U}}$. □

6.3 How To Jointly Choose Random Values

For the security of our schemes it is essential that the random values R and ρ_i, $0 \leq i \leq m$ for basic glitch protection and R_i and $\rho_{i,j}$ for window glitch protection are jointly chosen uniformly at random from $\mathbb{Z}_q^*$ by $\mathcal{U}$ and $\mathcal{V}$. The naive approach would be to use a coin flipping protocol for all of these values. For instance $\mathcal{U}$ may choose $x' \xleftarrow{r} \mathbb{Z}_q^*$ and sends a commitment $g^{x'}$ to V, V sends U a value $x'' \xleftarrow{r} \mathbb{Z}_q^*$ and $\mathcal{U}$ opens the commitment by sending x'. Now both parties compute $x = x' + x''$. The value x is jointly chosen. To save effort it is possible to negotiate only one value and generate all other values either using a PRF $F_{\cdot}(\cdot)$ or a hash function $H(\cdot)$. Let $Y(\cdot)$ be either $F_x(\cdot)$ or $H(x||\cdot)$. Then we generate R as $Y(m + 1)$ and ρ_i as $Y(i)$.

7 Additional Extensions

One advantage of our approach to periodic anonymous authentication is that its modular construction fits nicely with previous work [13, 16]. Thus, it is clear which parts of our system can be modified to enable additional features.

7.1 Weak Exculpability

Recall that *weak exculpability* allows an honest verifier (or group of verifiers) to prove in a sound fashion that the user with public key $pk_{\mathcal{U}}$ reused *some* token. This convinces everyone in the system that the user with $pk_{\mathcal{U}}$ is untrustworthy.

To implement weak exculpability, we need to define algorithm $\mathsf{VerifyViolator}$ and to slightly adapt the $\mathsf{IKeygen}$, $\mathsf{UKeygen}$, Show, and $\mathsf{Identify}$ algorithms. $\mathsf{IKeygen}'$ now also runs $\mathsf{Bilinear\_Setup}$, and the parameters for the bilinear map $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ are added to $pk_{\mathcal{I}}$. $\mathsf{UKeygen}'$ selects a random $sk_{\mathcal{U}} \in \mathbb{Z}_q^*$ and outputs $pk_{\mathcal{U}} = \mathbf{e}(g_1, g_2)^{sk_{\mathcal{U}}}$. In the Show' protocol, the double-spending tag is calculated as $E = g_1^{sk_{\mathcal{U}}} \cdot F_{(g_1,s)}(c(1, T, J))^R$. Consequently the value s_U, returned by $\mathsf{Identify}'$, is $g_1^{sk_{\mathcal{U}}}$ – which is secret information! Thus, the $\mathsf{VerifyViolator}$ algorithm is defined as follows: $\mathsf{VerifyViolator}(pk_{\mathcal{I}}, pk_{\mathcal{U}}, s_{\mathcal{U}})$ accepts only if $\mathbf{e}(s_{\mathcal{U}}, g_2) = \mathbf{e}(g_1^{sk_{\mathcal{U}}}, g_2) = pk_{\mathcal{U}}$. Intuitively, because $g_1^{sk_{\mathcal{U}}}$ is *secret* information, its release signals that this user misbehaved.

A subtle technical problem with this approach is that tag E is now set in a bilinear group $\mathbb{G}_1$, where DDH may be easy, and we need to ensure that the DY PRF is still secure in this group. Indeed, in groups where DDH is easy, the DY PRF is *not* secure. There are two solutions [13]: (1) make the XDH assumption, i.e., DDH is hard in $\mathbb{G}_1$, and continue to use the DY PRF, or (2) make the more general Sum-Free DDH assumption and use the CHL PRF [13], which works in groups where (regular) DDH is easy.

Theorem 7.1 *The above scheme provides* weak exculpability *under the Strong RSA, y-*DDHI *if $l_x \in O(\log k)$ or* SDDHI*, and either XDH or Sum-Free DDH assumptions.*

7.2 Strong Exculpability

Recall that *strong exculpability* allows an honest verifier (or group of verifiers) to prove in a sound fashion that the user with public key $pk_{\mathcal{U}}$ reused an e-token with TSN S.

For strong exculpability, we need to define VerifyViolation and to adapt the Show and the Identify algorithms. In Show$''$, the ZK proof of validity is transformed into a non-interactive proof, denoted Π, using the Fiat-Shamir heuristic [34]. The proof Π is added to the coin transcript, denoted τ. And Identify$''(pk_{\mathcal{I}}, S, \tau_1, \tau_2)$ adds both transcripts τ_1, and τ_2 to its output $s_{\mathcal{U}}$. (The function $\phi(s_{\mathcal{U}}) = pk_{\mathcal{U}}$ ignores the extra information.)

Thus, the VerifyViolation algorithm is defined as follows: VerifyViolation$(pk_{\mathcal{I}}, S, pk_{\mathcal{U}}, s_{\mathcal{U}})$ parses $\tau_1 = (E_1, R_1, \Pi_1)$ and $\tau_2 = (E_2, R_2, \Pi_2)$ from $s_{\mathcal{U}}$. Then, it checks that $\phi(s_{\mathcal{U}}) = pk_{\mathcal{U}}$ and that Identify$''(pk_{\mathcal{I}},\ S, \tau_0, \tau_2) = s_{\mathcal{U}}$. Next, it verifies both non-interactive proofs Π_i with respect to (S, R_i, T_i). If all checks pass, it accepts; else, it rejects.

A subtlety here is that, for these proofs to be sound even when the issuer is malicious, the group $\mathbf{G}'$ that is needed as a parameter for zero-knowledge proofs here must be a system parameter generated by a trusted third party, such that no one, including the issuer, knows the order of this group. So in particular, $\mathbf{G}'$ cannot be the same as $\mathbf{G}$ [21].

Theorem 7.2 *The above scheme provides* strong exculpability *under the Strong RSA, and y-DDHI if $l_x \in O(\log k)$ or SDDHI assumptions in the random oracle model with trusted setup for the group $\mathbf{G}'$.*

7.3 Tracing

We can extend our periodic n-times authentication scheme so that if a user reuses even one e-token, *all* possible TSN values she could compute using *any* of her dispensers are now publicly computable. We use the same IKeygen$'$, UKeygen$'$, Show$'$, and Identify$'$ algorithms as weak exculpability, slightly modify the Obtain protocol, and define a new Trace algorithm.

In UKeygen$'$, the user's keypair $(\mathbf{e}(g_1, g_2)^{sk_{\mathcal{U}}}, sk_{\mathcal{U}})$ is of the correct form for the bilinear ElGamal cryptosystem, where the value $g_1^{sk_{\mathcal{U}}}$ is sufficient for decryption. Now, in our modified Obtain$'$, the user will provide the issuer with a verifiable encryption [12] of PRF seed s under *her own* public key $pk_{\mathcal{U}}$. The issuer stores this tracing information in $R_{\mathcal{U}}$. When Identify$'$ exposes $g_1^{sk_{\mathcal{U}}}$, the issuer may run the following trace algorithm:

Trace$(pk_{\mathcal{I}}, pk_{\mathcal{U}}, s_{\mathcal{U}}, R_{\mathcal{U}}, n)$. The issuer extracts $g_1^{sk_{\mathcal{U}}}$ from $s_{\mathcal{U}}$, and verifies this value against $pk_{\mathcal{U}}$; it aborts on failure. The issuer uses $g_1^{sk_{\mathcal{U}}}$ to decrypt all values in $R_{\mathcal{U}}$ belonging to that user, and recovers the PRF seeds for *all* of the user's dispensers. For seed s and time t, all TSNs can be computed as $f_s(t, j) = F_{(\mathbf{e}(g_1, g_2), s)}(c(0, t, j))$, for all $0 \leq j < n$.

Theorem 7.3 *The above scheme provides* tracing of violators *under the Strong RSA, y-DDHI if $l_x \in O(\log k)$ or SDDHI, and either XDH or Sum-Free DDH assumptions.*

7.4 Dynamic Revocation

Implementing dynamic revocation requires modifying the Obtain and Show protocols in the basic scheme, and defining a new Revoke algorithm.

The mechanisms introduced in [16] can be used for revoking CL signatures. In an adjusted CL protocol for obtaining a signature on a committed value, the user obtains an additional witness $\mathbf{w} = \mathbf{v}^{e^{-1}}$, where $\mathbf{v}$ is the revocation public key and e is a unique prime which is part of the CL signature σ. In the CL protocol for proving knowledge of a signature, the user also proves knowledge of this witness. Violators with prime $\tilde{e}$ can be excluded by updating the revocation public key $\mathbf{v}$, such that $\mathbf{v}' = \mathbf{v}^{\tilde{e}^{-1}}$, and publishing $\tilde{e}$. While all non-excluded users can update their witness by computing function $f(e, \tilde{e}, \mathbf{v}', \mathbf{w}) = \mathbf{w}'$, without knowing the order of $\mathbf{G}$, this update does *not* work when $e = \tilde{e}$.

Thus, our e-token dispensers can be revoked by revoking their CL signature σ. Obtain''' is adapted to provide users with a witness $\mathbf{w}$ and to store the corresponding e as r_D. Show''' is adapted to update and prove knowledge of the witness. The $\mathsf{Revoke}(pk_{\mathcal{I}}, r_D)$ algorithm is defined as follows: Compute $\mathbf{v}' = \mathbf{v}^{r_D{}^{-1}}$ and publish it together with update information r_D. Additional details are in [16].

Theorem 7.4 *The above scheme provides* dynamic revocation *under the Strong RSA, and y-DDHI if $l_x \in O(\log k)$ or SDDHI assumptions.*

8 Acknowledgments

Part of Jan Camenisch's work reported in this paper is supported by the European Commission through the IST Programme under Contracts IST-2002-507932 ECRYPT and IST-2002-507591 PRIME. The PRIME projects receives research funding from the European Community's Sixth Framework Programme and the Swiss Federal Office for Education and Science. Part of Susan Hohenberger's work is supported by an NDSEG Fellowship. Markulf Kohlweiss is supported by the European Commission through the IST Programme under Contract IST-2002-507591 PRIME and partially by IWT SBO ADAPID. Anna Lysyanskaya is supported by NSF Grant CNS-0347661. Mira Meyerovich is supported by a U.S. Department of Homeland Security Fellowship and NSF Grant CNS-0347661. All opinions expressed in this paper are the authors' and do not necessarily reflect the policies and views of EC, DHS, and NSF.

References

[1] G. Ateniese, J. Camenisch, M. Joye, and G. Tsudik. A practical and provably secure coalition-resistant group signature scheme. In M. Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, p. 255–270, 2000.

[2] G. Ateniese, K. Fu, M. Green, and S. Hohenberger. Improved Proxy Re-encryption Schemes with Applications to Secure Distributed Storage. In *NDSS*, p. 29–43, 2005.

[3] M. H. Au, W. Susilo, and Y. Mu. Constant-size dynamic k-TAA. In *Fifth Conference on Security and Cryptography for Networks (SCN 2006)*, volume 4116 of *Lecture Notes in Computer Science*. Springer, 2006.

[4] L. Ballard, M. Green, B. de Medeiros, and F. Monrose. Correlation-Resistant Storage. Johns Hopkins University, CS Technical Report # TR-SP-BGMM-050705. `http://spar.isi.jhu.edu/~mgreen/correlation.pdf`, 2005.

[5] N. Barić and B. Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *EUROCRYPT '97*, volume 1233, p. 480–494, 1997.

[6] D. Boneh and X. Boyen. Short signatures without random oracles. In *EUROCRYPT 2004*, volume 3027 of LNCS, p. 56–73, 2004.

[7] D. Boneh, X. Boyen, and H. Shacham. Short group signatures using strong Diffie-Hellman. In *CRYPTO*, volume 3152 of LNCS, p. 41–55, 2004.

[8] D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. In J. Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, p. 213–229, 2001.

[9] F. Boudot. Efficient proofs that a committed number lies in an interval. In *EUROCRYPT '00*, volume 1807 of *LNCS*, p. 431–444, 2000.

[10] S. Brands. *Rethinking Public Key Infrastructure and Digital Certificates— Building in Privacy*. PhD thesis, Eindhoven Inst. of Tech. The Netherlands, 1999.

[11] E. Brickell, P. Gemmel, and D. Kravitz. Trustee-based tracing extensions to anonymous cash and the making of anonymous change. In *SIAM*, p. 457–466, 1995.

[12] J. Camenisch and I. Damgård. Verifiable encryption, group encryption, and their applications to group signatures and signature sharing schemes. In T. Okamoto, editor, *ASIACRYPT '00*, volume 1976 of *LNCS*, p. 331–345, 2000.

[13] J. Camenisch, S. Hohenberger, and A. Lysyanskaya. Compact E-Cash. In *EUROCRYPT*, volume 3494 of LNCS, p. 302–321, 2005.

[14] J. Camenisch, S. Hohenberger, and A. Lysyanskaya. Balancing accountability and privacy using e-cash. In *SCN (to appear)*, 2006.

[15] J. Camenisch and A. Lysyanskaya. Efficient non-transferable anonymous multi-show credential system with optional anonymity revocation. In B. Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, p. 93–118. Springer Verlag, 2001.

[16] J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In M. Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, p. 61–76. Springer Verlag, 2002.

[17] J. Camenisch and A. Lysyanskaya. A signature scheme with efficient protocols. In *SCN 2002*, volume 2576 of *LNCS*, p. 268–289, 2002.

[18] J. Camenisch and A. Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *CRYPTO 2004*, volume 3152 of *LNCS*, p. 56–72, 2004.

[19] J. Camenisch, U. Maurer, and M. Stadler. Digital payment systems with passive anonymity-revoking trustees. In E. Bertino, H. Kurth, G. Martella, and E. Montolivo, editors, *Computer Security — ESORICS 96*, volume 1146 of *LNCS*, p. 33–43. Springer Verlag, 1996.

[20] J. Camenisch and M. Michels. Proving in zero-knowledge that a number n is the product of two safe primes. In *EUROCRYPT '99*, volume 1592, p. 107–122, 1999.

[21] J. Camenisch and M. Michels. Separability and efficiency for generic group signature schemes. In *CRYPTO '99*, volume 1666 of *LNCS*, p. 413–430, 1999.

[22] J. Camenisch and M. Stadler. Efficient group signature schemes for large groups. In B. Kaliski, editor, *CRYPTO '97*, volume 1296 of *LNCS*, p. 410–424. Springer Verlag, 1997.

[23] A. Chan, Y. Frankel, and Y. Tsiounis. Easy come – easy go divisible cash. In *EUROCRYPT '98*, volume 1403 of *LNCS*, p. 561–575, 1998.

[24] D. Chaum. Blind signatures for untraceable payments. In *CRYPTO '82*, p. 199–203. Plenum Press, 1982.

[25] D. Chaum. Blind signature systems. In *CRYPTO '83*, p. 153–156. Plenum, 1983.

[26] D. Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, Oct. 1985.

[27] D. Chaum, A. Fiat, and M. Naor. Untraceable electronic cash. In *CRYPTO '90*, volume 403 of *LNCS*, p. 319–327, 1990.

[28] D. Chaum and E. van Heyst. Group signatures. In D. W. Davies, editor, *EUROCRYPT '91*, volume 547 of *LNCS*, p. 257–265. Springer-Verlag, 1991.

[29] S. Chawla, C. Dwork, F. McSherry, A. Smith, and H. Wee. Toward privacy in public databases. In J. Kilian, editor, *Second Theory of Cryptography Conference*, volume 3378 of *Lecture Notes in Computer Science*, p. 363–385. Springer, 2005.

[30] I. Damgard, K. Dupont, and M. O. Pedersen. Unclonable group identification. Cryptology ePrint Archive, Report 2005/170, 2005. `http://eprint.iacr.org/2005/170`.

[31] I. B. Damgård. Payment systems and credential mechanism with provable security against abuse by individuals. In S. Goldwasser, editor, *CRYPTO '88*, volume 403 of *LNCS*, p. 328–335. Springer Verlag, 1990.

[32] Y. Dodis. Efficient construction of (distributed) verifiable random functions. In *Public Key Cryptography*, volume 2567 of *LNCS*, p. 1–17, 2003.

[33] Y. Dodis and A. Yampolskiy. A Verifiable Random Function with Short Proofs an Keys. In *Public Key Cryptography*, volume 3386 of LNCS, p. 416–431, 2005.

[34] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, volume 263 of LNCS, p. 186–194, 1986.

[35] E. Fujisaki and T. Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In *CRYPTO '97*, volume 1294 of *LNCS*, p. 16–30, 1997.

[36] S. D. Galbraith. Supersingular curves in cryptography. In C. Boyd, editor, *ASIACRYPT*, volume 2248 of LNCS, p. 495–513, 2001.

[37] S. Jarecki and V. Shmatikov. Handcuffing big brother: an abuse-resilient transaction escrow scheme. In *EUROCRYPT*, volume 3027 of *LNCS*, p. 590–608, 2004.

[38] A. Kiayias, M. Yung, and Y. Tsiounis. Traceable signatures. In C. Cachin and J. Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, p. 571–589. Springer, 2004.

[39] A. Lysyanskaya. *Signature Schemes and Applications to Cryptographic Protocol Design*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, Sept. 2002.

[40] A. Lysyanskaya, R. Rivest, A. Sahai, and S. Wolf. Pseudonym systems. In H. Heys and C. Adams, editors, *Selected Areas in Cryptography*, volume 1758 of *LNCS*, 1999.

[41] N. McCullagh and P. S. L. M. Barreto. A new two-party identity-based authenticated key agreement. In *CT-RSA*, volume 3376 of LNCS, p. 262–274, 2004.

[42] V. I. Nechaev. Complexity of a determinate algorithm for the discrete logarithm. *Mathematical Notes*, 55:165–172, 1994.

[43] L. Nguyen and R. Safavi-Naini. Dynamic k-times anonymous authentication. In *ACNS 2005*, number 3531 in LNCS, p. 318–333. Springer Verlag, 2005.

[44] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO '92*, volume 576 of *LNCS*, p. 129–140, 1992.

[45] M. Scott. Authenticated ID-based key exchange and remote log-in with simple token and PIN number. Available at http://eprint.iacr.org/2002/164, 2002.

[46] V. Shoup. Lower bounds for discrete logarithms and related problems. In *EUROCRYPT*, LNCS, p. 256–266, 1997. Update: `http://www.shoup.net/papers/`.

[47] L. Sweeney. k-anonymity: a model for protecting privacy. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10(5):557–570, 2002.

[48] I. Teranishi, J. Furukawa, and K. Sako. k-times anonymous authentication (extended abstract). In *Asiacrypt 2004*, volume 3329 of *LNCS*, p. 308–322. Springer Verlag, 2004.

[49] I. Teranishi and K. Sako. -times anonymous authentication with a constant proving cost. In M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, editors, *Public Key Cryptography*, volume 3958 of *Lecture Notes in Computer Science*, p. 525–542. Springer, 2006.