FFI Forsvarets forskningsinstitutt

FFI-NOTAT

Eksternnotat 22/00178

Anonymous tokens

– implementation and development

Authors
Teodor Dahl Knutsen, Tallak Manum, Martin Strand
Project number 1614
18 January 2022

Approvers
Raymond Haakseth, *Research Manager*; Source code and license are approved by Raymond Haakseth
The document is electronically approved and therefore has no handwritten signature.

Keywords
kryptografi, kryptologi, autentisering, åpen kildekode, datainnsamling

Summary
Anonymous tokens can be used to provide one-time authentication for a service without revealing one's identity, and has been used as a CAPTCHA replacement and to provide additional anonymity in the Norwegian COVID-19 contact tracing app, and more.
We demonstrate the practicality of implementing anonymous tokens with public metadata. Further, the protocols are extended to include hidden public metadata, and improved batch verification across different metadata. An attack on previously suggested batch verification is also demonstrated.
We envision further applications, for instance for services where the identity of some data submitting operative is more sensitive than the data it submits.

E

Contents

Preface

Teodor Dahl Knutsen and Tallak Manum, who have contributed the majority of the work presented here, were summer interns at FFI during the summer of 2021. For this season, FFI received over 1,000 applications from some of the brightest students in Norway, and it is truly inspiring to work with the approximately 70 successful applicants.

In cooperation, Teo and Tallak have studied the whole stack, ranging from algebraic geometry, through cryptographic adversarial models, to practical implementation, some side channel considerations, and also the very difficult task of communicating these complicated topics to a non-technical audience. It is my privilege to thank them for their great work.

I would also use the opportunity to thank Tjerand Silde for his cooperation on this topic, as well as Entur, Bekk and their own summer interns who examined the same topic from a different perspective, and shared their insights with us.

Kjeller, 18 January 2022
Martin Strand

1 Introduction

How can one provide proof of authorisation to an action or resource without disclosing one's identity? This question is relevant on a number of examples:

- Cash is anonymous: If you boarded a bus and paid for the ticket in cash, the ticket would not be easily traceable to you. However, the fact that you could present cash authorised you to use the service.
- Many privacy-minded people are using services like The Onion Router (Tor) to browse the internet. However, as the service is also extensively used by malicious computer networks, websites often requier Tor users to prove that they are humans before using their service. This causes a usability issue, as users repeatedly need to complete CAPTCHA challenges.
- App creators want to collect telemetry data from their products. However, as this also carries the risk of surveillance, the data should be anonymous. How can creators defend against malicious users who would try to inject too much or erroneous data into the system, for example in order to confuse content generation algorithms?
- Elections need to be private, but they should also ensure that only eligible voter cast a ballot, and at most one ballot per voter should be tallied.

These questions have prompted researchers to propose privacy-preserving solutions while maintaining a strong guarantee for the functionality.

In this work we present some previous work on these issues, and provides an implementation of new protocols by Silde and Strand [7]. We have four central contributions:

- Finally, we provide implementations of two of Silde and Strand's protocols, along with a sample application and a physical demonstration of this. The code is available on Github[1], and its usage is described briefly in the appendices.
- Both Silde and Strand and Tyagi et al. [8] mention a potential linking attack if the protocol is instantiated with a certain type of bilinear pairings. We have verified that the attack does indeed *not* work. See Section 3.2 for details.
- We introduce hidden metadata, decided by the token holder. This data is completely hidden from the token issuer and verifier until the holder decides to disclose it; yet the holder is committed to its choice as long as the underlying hash function is secure against second preimage attacks. Our technique is further described in Section 4.1.
- We identified a bug in the writeup of Silde and Strand which would allow a user to submit arbitrarily many forged tokens as long as they would be verified in batch. In Section 4.2, we propose a modification to the batch verification procedure to rectify this.

This note does not aim to be a complete survey of anonymous credentials; we refer to the cited works for further details.

1.1 Motivation

Consider a scenario with three parties: a signer, a user and a verifier. The user wants a piece of paper from the signer, proving to the verifier that the signer has granted the user access to some resource. This signer wants this piece of paper to be unique, such that the user may not copy this

[1] `https://github.com/FFI-no/Paper-anonymous-tokens`

piece of paper and give it to their friends. However, the user wishes that even if the signer and verifier conspires together, they will not be able to identify the user when they redeem this access that was granted by the signer.

The user selects a random, unique number and writes this down on a piece of paper. On this piece of paper, they write down some more information, metadata, e.g. the resource they wish to receive access to. This is their token. The user puts this token in an envelope made out of carbon paper and seals it. On another piece of paper the user writes down the metadata again. This metadata and the envelope is handed over to the signer, in addition to identifying themselves to prove to the signer that they should have access to this resource from the metadata. If the signer agrees with this, they write the metadata on the envelope, in addition to their signature, which should be copied to the paper token inside by the carbon paper. The envelope and the metadata is handed back to the user. The user will discard the envelope and now has a unique, signed token granting access to some resource.

With these pieces of paper, the user approaches the verifier, requesting to redeem the token in exchange for access to some resource. The verifier sees the signed token and verifies that the signature is valid and that the metadata provided matches the resource requested. If the token is valid, and not previously used the user may get access to the resource. They write down the number from the token in their logbook in order to avoid reuse of the same token.

Now, remember that only the signer knows the identity of the user, but the signer does not know the unique number of the token. The verifier does not know the identity of the user, but they know the number of the token. The verifier and the signer are therefore not able to cooperate to find some correlation between the identity of the user and the number on the token.

Inside the envelope, the user may in addition to the number write some more information that they want to hide from the signer but show the verifier. In the context of electronic voting, this could be their voting preferences.

1.2 Previous work

Anonymous tokens were originally introduced by Davidson et al. [2] as *Privacy pass* in order to let Tor users prove that they are humans without the use of CAPTCHAs for each instance. Instead, users would solve a CAPTCHA problem towards a Cloudflare server, which issues a number of tokens to be modified and stored in the browser. When visiting a website hosted by Cloudflare, the browser presents a fresh token, and the user is automatically accepted. The tokens are not traceable back to the issuance phase.

Facebook has adopted the same idea for collecting WhatsApp telemetry anonymously [3], but also added a key rotation mechanism to aid abuse protection.

Kreuter et al. [4] expanded the idea in a different direction by adding suppport for a private metadata bit. The signer may inject a single bit of information into the token, such that it remains unaccessible to the user, but readable to the token verification service. Notice that this reduces anonymity by one bit, which allows the service to separate users into two groups, for example "trusted" and "untrusted" users.

Moe, Silde and Strand [5] reimplemented Privacy Pass for use in the COVID-19 digitial contact tracing app Smittestopp. During their work and its application, they experienced that also anonymous tokens can benefit from having date stamps. Silde and Strand [7] provided a new construction of anonymous tokens with both private and public metadata. Independently, Tyagi et al. [8] have presented the same construction, along with a complete security proof.

2 Background

We start by briefly introducing some necessary background theory. The goal of this section is to give a brief overview for someone unfamiliar with the fundamentals of cryptography. While we do not expect the reader to be an expert in cryptography, we assume a working knowledge of algebra for the remainder of this text.

2.1 Computational problems

Cryptography is frequently based on hard computational problems. The idea is that one can prove that any successful attacker against the cryptographic scheme will also be able to break a given instance of the problem of a certain size. Thus, the analysis of several cryptosystems can be reduced to studying a single, clearly stated problem.

2.1.1 The discrete log problem

Given a multiplicative group $\mathbb{G}$ of prime order p and an interger n, define g^n as n copies of g multiplied together by the group operation. We then define the discrete logarithm of two group elements $\log_a b$ as an integer n such that $a^n = b$. Finding discrete logarithms is a problem that is considered hard in some groups and this is fundamental to the system used for this project.

2.1.2 The Diffie-Hellman problem

The Diffie-Hellman problem can briefly be described as understanding the relation between the group elements g, g^a, g^b, g^{ab}, and is the basis for many cryptosystems. It plays a central role in this system as well.

Computational Diffie-Hellman The computational Diffie-Hellman problem (CDH) can be stated as “Given three group elements (g, g^a, g^b) find g^{ab}.”

Decisional Diffie-Hellman The decisional Diffie-Hellman problem (DDH) can be stated as follows “Given four group elements (g, g^a, g^b, g^z) verify whether $z = ab$.”

Gap Diffie-Hellman A gap Diffie-Hellman group is a group in which the decisional Diffie-Hellman problem is easy to solve, but the computational Diffie-Hellman problem remains hard. A gap Diffie-Hellman group is useful for signature schemes since we can publish a public key g^k and then when someone wants to verify that some message tuple (g^a, g^z) is signed by our private key k, they can verify that (g, g^a, g^k, g^z) is a Diffie-Hellman tuple, that is, that $z = ka$. However, since computational Diffie-Hellman is still hard it is difficult for someone to self-sign their message, even if they know the public key.

2.1.3 Elliptic curves

An elliptic curve E is the set of points that satisfy an equation of the form $Y^2 = X^3 + aX + b$. With these points, one can define “addition” of two points, see Figure 2.1. Find the line that intersects both of the given two points. It will also intersect a third point on the curve. Mirror this third point

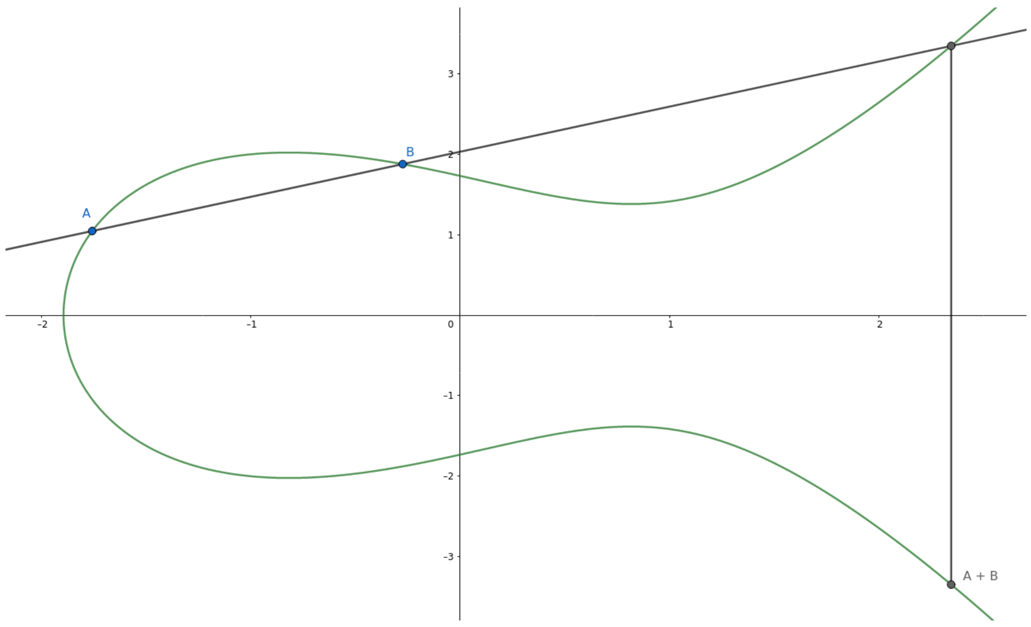


Figure 2.1 Finding the sum of the points A and B on the elliptic curve $Y^2 = X^3 - 2X + 3$.

about the x-axis to get the sum. If the line is vertical, we say that it intersects $\mathcal{O}$, the “point at infinity”. If you add $\mathcal{O}$ to any point A, you get a vertical line and the sum is A.

Adding a point to itself is finding the tangent at that point and see where it intersects the elliptic curve. Mirror this about the x-axis, and the resulting point is the double. These operations define a group law. It is customary to write the group operation on elliptic curves additively. Addition of a point to itself n times is denoted by the map $[n] : E \longrightarrow E$. Given two points P and $Q = [n]P$, the discrete logarithm problem is to find n. This is hard on some elliptic curves.

The figure shows the group law for an elliptic curve over the reals. We will use curves defined over finite fields, but the formulas one can derive from the above description remain valid and unambiguous.

2.2 Pairings

Let $\mathbb{G}_1, \mathbb{G}_2$ be groups written additively and let $\mathbb{G}_T$ be a multiplicative group. A bilinar pairing is a map $\mathbb{G}_1 \times \mathbb{G}_2 \longrightarrow \mathbb{G}_T$ such that

$$\begin{aligned} \hat{e}(P + P', Q) &= \hat{e}(P, Q) \cdot \hat{e}(P', Q) \\ \hat{e}(P, Q + Q') &= \hat{e}(P, Q) \cdot \hat{e}(P, Q') \\ \hat{e}(P, P) &\neq 1 \end{aligned}$$

holds for all P, P', Q, Q', n. Notice in particular that $\hat{e}([n]P, Q) = \hat{e}(P, [n]Q) = \hat{e}(P, Q)^n$

There exist efficiently computable such functions for certain subgroups of certain elliptic curves. These pairings allows us to efficiently verify whether $c = a \cdot b$, given four group elements $(G_2, [a]G_1, [b]G_2, [d]G_1)$, by verifying the equality

$$\hat{e}([a]G_1, [b]G_2) = \hat{e}([c]G_1, G_2).$$

That is, we have a generalized version of the decisional Diffie-Hellman problem that is easy to solve. If the generalized computational Diffie- Hellmann is still dificult to solve then we have a generalized version gap Diffie-Hellmann group.

2.3 Non-Interactive Zero-Knowledge Proofs

Non-interactive zero-knowledge proofs (NIZK) are protocols which one can use to generate a convincing argument that a statement is true without revealing anything about its nature.

In this project we use one such protocol to demonstrate that $\log_W T = \log_K G$. The secret necessary to demonstrate this is the common value of these logarithms. Silde and Strand [7] goes more into detail of this standard proof. A batched variant is also discussed in further detail by Davidson et al. [2]. Using NIZK means that even if we do not have a gap Diffie-Hellman group we can still be convinced that we have received a correctly signed token from the signer.

3 Protocols for anonymous tokens

We now present Silde and Strand's protocols [7], primarily focusing on the parings-based instantiation. The protocols are more generally known as *verifiable (partially) oblivious pseudo-random functions*. Pseudo-random functions (PRF) produce output that will look random unless one knows secret key material. Oblivious PRFs are protocols that can compute a PRF without any of the parties learning the other party's secret input, and one one of the parties learning the output of the function. Verifiable oblivious PRFs (VOPRF) guarantee that correct input has been used. The output from the VOPRF is used as the anonymous token, since the user's private input is unknown to the issuer.

The protocols use two mechanisms for creating this verifiability: The first set of protocols use non-interactive zero-knowledge proofs in order to enable an issuer to prove that the correct private key was used to generate the function output. However, this proof cannot be updated by the receiver, and so the resulting token can only be verified by a party holding the private key.

Seeing this as a disadvantage, Silde and Strand also devised a VOPRF instantiation that allows for verification without using zero-knowledge proofs, instead using bilinear pairings. This protocol is presented below, in its batch verification variant.

3.1 Pairing-based protocol

The protocol is described in Figure 3.1[2]. Let $\mathtt{H}_1 : \{0,1\}^* \to \mathbb{G}_1$, $\mathtt{H}_m : \{0,1\}^* \to \mathbb{Z}_p^*$ be hash functions. Let (t_i), (md_i) be lists of n elements over token seeds and their respective metadata, and let K be the public key. Let $a \leftarrow\!\!\$\ A$ denote the sampling of a value from the set A. Unless otherwise stated, the sampling is assumed to be uniformly random.

[2]Silde and Strand's original draft defined $T := \mathtt{H}_1(t\|\mathsf{md})$. This has later been revised to $T := \mathtt{H}_1(t)$. For the purpose of this work, we use the former variant.

Attestation

User $((t_i), (\mathsf{md}_i), K)$		**Signer**$((\mathsf{md}_i), k)$
$(T_i := \mathtt{H}_1(t_i \Vert \mathsf{md}_i))$		
$(r_i \leftarrow\!\!\$\; \mathbb{Z}_p^*)$		
$(T_i' := [r_i^{-1}]T_i)$	$\xrightarrow{(T_i')}$	
		$(d_i := \mathtt{H}_m(\mathsf{md}_i))$
		$(e_i := (d_i + k)^{-1})$
$(d_i := \mathtt{H}_m(\mathsf{md}_i))$	$\xleftarrow{(W_i')}$	$(W_i' := [e_i]T_i')$

```
$(W_i := [r]W_i')$
**if** $\hat{e}\left(\sum_i W_i, K\right) \overset{?}{=} \hat{e}\left(\sum_i (T_i - [d_i]W_i), G_2\right)$ :
    **return** $(W_i)$
**else**
    **return** $\perp$
```

Verification

User $((t_i, \mathsf{md}_i, W_i))$		**Verifier**(K)
	$\xrightarrow{(t_i, \mathsf{md}_i, W_i)}$	$(r_i \leftarrow\!\!\$\; \mathbb{Z}_p^*)$
		$(d_i := \mathtt{H}_m(\mathsf{md}_i))$
		$(W_i' := [r_i]W_i)$
		$(T_i' := [r_i]\mathtt{H}_1(t_i \Vert \mathsf{md}_i) - [d_i]W_i')$
		return $\hat{e}\left(\sum_i r_i W_i', K\right) \overset{?}{=} \hat{e}\left(\sum_i r_i T_i', G_2\right)$

Figure 3.1 The protocol for the pairing-based scheme.

3.2 Linking attacks

Anonymous tokens should make it impossible to link the user between the two phases of the protocol: even if the signer colludes with the verifier, it should remain impossible to identitfy the user using only the tokens.

Tyagi et al. [2] mention a linking attack against the protocol but without further elaboration. Assuming we have some computable homomorphism $\phi : \mathbb{G}_1 \longrightarrow \mathbb{G}_2$, a linking attack is described in an earlier version of [7]. We interpret the attack described as follows: given a randomized valid token pair (T', W') and a unrandomized valid token pair (T, W) that share metadata, check if the first pair is a randomization of the second by checking the equality $\hat{e}(W, \phi(T')) = \hat{e}(W', \phi(T))$. Given that the pairs of tokens are the same pair but randomized there exists an r such that $(rT, rW) = (T', W')$. This then implies that the equation must hold true since $\hat{e}(W, \phi(T')) = \hat{e}(W, \phi(rT)) = \hat{e}(W, r\phi(T)) = \hat{e}(rW, \phi(T)) = \hat{e}(W', \phi(T))$.

However, in our interpretation of this attack, it seems to not be effective, since given a pair (T', W') and (T, W) that are not necessarily related there will always exist an $r = \log_T T'$ since the group is cyclic of prime order. Then assuming (T, W) and (T', W') are valid token pairs which share metadata we can deduce $[r]W = [re]T = [e][r]T = [e]T' = W'$. Hence we are in the same situation as before, i.e we have a r such that $([r]T, [r]W) = (T', W')$ and therefore the test will tell you that any two token pairs are linked.

We have managed to identify one possible linking attack, assuming that someone is reusing the same constant for randomization- Then we can link two different token pairs randomized with the same constant and simultaneously link the randomized token pair to its unrandomized token pair, also assuming that we have some computable homomorphism $\phi : \mathbb{G}_1 \longrightarrow \mathbb{G}_2$.

Given a set of signed randomized token pairs $([r_i]T_i, [r_i]W_i)_{i \in [0,n]}$ and a corresponding set of signed unrandomized token pairs $(T_j, W_j)_{j \in [0,n]}$ we want to figure out which token are randomized with the same r. We choose two randomized token pairs $([r_k]T_k, [r_k]W_k)$ and $([r_m]T_m, [r_m]W_m)$ and two unrandomized pairs (T_s, W_s) and (T_p, W_p). We then verify the equality

$$\hat{e}([r_k]T_k, \phi(W_p)) \overset{?}{=} \hat{e}([r_m]T_m, \phi(W_s))$$

We rewrite each side of th equation using $t_i = \log_{G_1} T_i$

$$\hat{e}([r_k]T_k, \phi(W_p)) = \hat{e}([r_k][t_k]G_1, \phi([et_p]G_1)) = \hat{e}(G_1, \phi(G_1))^{r_k t_k e t_p}$$
$$\hat{e}([r_m]T_m, \phi(W_s)) = \hat{e}([r_m][t_m]G_1, \phi([et_s]G_1)) = \hat{e}(G_1, \phi(G_1))^{r_m t_m e t_s}$$

Verifying the above equality becomes equivalent to verifying the following equality

$$\hat{e}(G_1, \phi(G_1))^{r_k t_k e t_p} \overset{?}{=} \hat{e}(G_1, \phi(G_1))^{r_m t_m e t_s}$$

which is equivalent to verifying

$$r_k t_k t_p \overset{?}{=} r_m t_m t_s$$

up to the order of $\mathbb{G}_T$.

Observe that if $s = k$, $m = p$ and $r_k = r_m$ this equation holds. That is, if $([r_k]T_k, [r_k]W_k)$ is a randomized version of (T_s, W_s) and $([r_m]T_m, [r_m]W_m)$ is a randomized version of (T_s, W_s) and these two have been randomized by the same $r = r_k = r_m$ and the equality will hold. Note that if not all the three requirements above holds then the equation will not be satisfied except with very small probability.

This attack is then able to not only tell us if someone has used the same r for two different pairs of tokens, but simultaneously will link the unrandomized token pairs to their respective randomized token pairs. This underlines a long-standing advise in cryptography: Never reuse randomness.

4 Results

We now report on the outcome of the summer project. We have analysed the protocols of Silde and Strand, as well as implemented them with the addition of new features. We start by describing the new features, and then return to benchmarking results from our implementation.

4.1 Hidden public metadata

It is possible to embed metadata hidden from the signer, but visible for the user and the verifier. When generating a token, the user picks a token seed t and some hidden metadata hmd. They calculate the new token seed as $t' = H(t\|\mathsf{hmd})$. Then, the unsigned token is calculated as $T = H_1(t'\|\mathsf{md})$. The verifier requires only the initial token seed, the hidden metadata and the metadata.

4.2 Batched signing and verification

In order to reduce computational cost, one can use a method for verifying a batch of tokens all at once. This could be useful in both the attestation protocol and in the redemption protocol. For the attestation, the user may get several tokens signed, and will verify that the batch is a valid batch. In the redemption protocol, a verifier could receive a large number of tokens from one or more users, and knows that attempts to forge a token are relatively uncommon.

For this purpose it seems useful to have a way of batch verifying tokens where they do not necessarily share metadata if e.g you want to verify a batch of buss tickets with metadata specifying at which specific times at which they are valid in the metadata.

4.3 Verification with metadata

Given a token pair (T, W) of signed tokens with metadata d, we want to verify that this is signed with the key $(d+k)^{-1}$. We can reduce this to verifying that the token pair (T', W) is a signed token pair without metadata, where $T' = T - [d]W$. This is then a correctly signed token pair if the original pair was correctly signed due to the following equation:

$$\log_W T' = \log_W T - [d]W = \log_W T - \log_W [d]W = k + d - d = k$$

If (T', W) is a correctly signed pair of tokens without metadata then (T, W) is a correctly signed pair of tokens with metadata d since

$$T = T' + [d]W = [k]W + [d]W = [k+d]W$$

This means that our original pair of tokens with metadata were correctly signed if and only if the new pair of tokens without metadata was correctly signed.

This trick simplifies batch verification of tokens with different metadata, since we can reduce it to verifying a batch of tokens where they all share metadata $d = 0$, that is, they have no metadata.

4.3.1 Batched NIZK

In the NIZK-version of the protocol, batch verification takes two different forms based on whether it is the user or the verifier that performs it. The batched NIZK proof is a slight tweak to the batching described in [2] by using the verification across different metadata discussed in the last section. The proof is the proof of the following statement,

$$\log_{\sum_i [r_i] W_i} \sum_i [r_i](T_i - [\mathtt{H}_m(\mathsf{md}_i)]W_i) = \log_K G,$$

where r_i is a random number generator output by $H(T_i, W_i, K, G)$ and $\mathtt{H}_m : \{0,1\}^* \to \mathbb{Z}_p^*$ is a hash function.

For verification, we assume that the verifier has access to the private key and simply reconstructs the signed tokens to see if it is a match. We have found no way of making it more efficient as it is already very effective.

4.3.2 Attack on previously suggested batch verification for pairing

An earlier version of Silde and Strand [7] suggested a batch verification

$$\hat{e}\left(\sum_i W_i, U\right) = \hat{e}\left(\sum_i T_i, G_2\right) \tag{4.1}$$

for the pairing based protocol where all tokens share metadata. If we assume the verifier uses this batch verification, a user that is allowed to get one signed token may create a batch of n valid tokens.

The adversary generates n unsigned tokens T_i and calculate the sum $\overline{T} = \sum_i T_i$. He uses this $\overline{T}$ in the signature protocol to get $\overline{W}$. He picks $n-1$ curve points W_i such that $O = \sum_i W_i$. The adversary now has points such that

$$\hat{e}\left(\overline{W} + \sum_i W_i, U\right) = \hat{e}\left(\sum_i T_i, G_2\right).$$

This problem may be remedied by a standard procedure of using randomized linear combinations. But, first note that we only need to do this for the special case with no metadata due to the earlier reduction.

4.3.3 Pairing batch verification with arbitrary linear combination

In order to fix the aforementioned attack we will employ linear combinations. This is done by generating a random scalar r_i for each token pair and then using batch verification on the sets $[r_i]T_i$, $[r_i]W_i$. The corrected version of 4.1 is then

$$\hat{e}\left(\sum_i [r_i]W_i, K\right) = \hat{e}\left(\sum_i [r_i]T_i, G_2\right)$$

Lemma 4.3.1. *Assume we have received n pairs of (T, W) non-trivial tokens (i.e. $T \neq O$) claimed to be valid signed token pairs without metadata. Then if not all tokens are individually correctly signed (i.e some $W_i \neq [k]T_i$) we have a $1 - 1/\mathrm{ord}(\mathbb{G}_1)$ probability of the batch verify with linear combination test giving a false positive. Furthermore, the test will never give a false negative.*

Proof. Note that the groups we are working over are cyclic. Fix generators G_1 and G_2. Also note that $\mathbb{G}_1$ has prime order and therefore every element, except the identity, is a generator and therefore every discrete log with any base, except identity, exists.

Now we define $t_i = \log_{G_1} T_i$ and $w_i = \log_{T_i} W_i$, i.e $[t_i]G_1 = T_i$ and $[w_i t_i]G_1 = [w_i]T_i = W_i$. We now use this to manipulate and expand the expression we are verifying.

$$\hat{e}\left(\sum_i [r_i]T_i, G_2\right) = \hat{e}\left(\sum_i [r_i t_i]G_1, G_2\right) = \hat{e}\left(G_1, G_2\right)^{\sum_i r_i t_i}$$

$$\begin{aligned}\hat{e}\left(\sum_i [r_i]W_i, K\right) &= \hat{e}\left(\sum_i [r_i w_i t_i]G_1, [k]G_2\right) \\ &= \hat{e}\left([k \sum_i r_i w_i t_i]G_1, G_2\right) \\ &= \hat{e}\left(G_1, G_2\right)^{k \sum_i r_i w_i t_i}\end{aligned}$$

The verification then becomes equivalent to verifying the following expression.

$$\hat{e}\left(G_1, G_2\right)^{k \sum_i r_i w_i t_i} = \hat{e}\left(G_1, G_2\right)^{\sum_i r_i t_i}$$

Which, considering we work in the cyclic subgroup generated by $\hat{e}(G_1, G_2)$ for which we know $\mathrm{ord}(\hat{e}(G_1, G_2)) = \mathrm{ord}(G_1) = p$ becomes equivalent to verifying

$$\sum_i k r_i w_i t_i = \sum_i r_i t_i \pmod{p}$$

$$\Updownarrow$$

$$\sum_i k r_i w_i t_i - r_i t_i = \sum_i \left(k w_i - 1\right) t_i r_i = 0 \pmod{p}$$

Observe that this equation then, given (T, W) and metadata, becomes a linear equation from $\mathbb{Z}_p^n \to \mathbb{Z}_p$ with the list (r_i) as the variable. The equation is satisfied if and only if (r_i) is in the kernel of the transformation. Note that if all tokens are correctly signed then all coefficients are zero, giving a trivial morphism meaning that every (r_i) vector is in the kernel, and therefore we can't have a false negative.

Given that all pairs are non trivial, i.e $t_i \neq 0$, and that some token is not correctly signed, i.e $w_i \neq k^{-1}$, the transformation is non-trivial hence the kernel is of dimension $n - 1$. The probability of a uniformly randomly chosen (r_i) being in the kernel is then $|\mathbb{Z}_p^{n-1}|/|\mathbb{Z}_p^n| = 1/p$. This is then the probability of accepting a batch of tokens where at least one token is not correctly signed. □

4.4 Benchmarks

All benchmarks are run on an Intel Core i7-5600U CPU at 2.60 GHz clockspeed.

	NIZK	Pairing
Generate	117	117
Randomize	88	752
Sign Randomized	208	557
Unrandomize and Verify	–	–
Verify	75	6375

Table 4.1 Comparison of the two protocols on individual tokens. All the times listed are in microseconds. These are the primitives in the token engine. Was not able to benchmark the unrandomize and verify function, since it will consume the tokens and the benchmarker was not able to run it multiple times.

The pairing based protocol is slower than the NIZK-based protocol, as Table 4.1 shows. We may see that the generation is the same for both protocols. The pairing based is slower than NIZK-based in the randomize function. They both do the same; hash to curve, picking a random scalar and inverting it, and scalar multiplication on a curve-point. The difference is possibly partly because the elliptic curve used in the NIZK implementation is optimized to be fast. The NIZK-based protocol is also faster in *sign randomized*. Notice that the NIZK-based protocol both signs the point and creates a proof, while the pairing implementation only signs the point. The difference in the verification is obvious, since calculating the pairing is slow.

	NIZK	NIZK batch	Pairing	Pairing batch
Generate and Sign	62.2	49.6	820.8	207.8
Verify	7.5	7.5	635.9	126.0

Table 4.2 Runtimes for generation and signing, and verification of 100 tokens, compared to the runtime of the same, using the batched variant of the protocols. All the numbers are in milliseconds.

There is no significant difference between NIZK with and without batching, as seen in Table 4.2. For the signing, batching produces one proof for the batch, whereas doing it individually will generate one proof per token. The batching variant could therefore create less network traffic.

For the pairing based protocol, batching is a significant improvement.

5 Applications

Before discussing the concrete protocols in more detail, let us explore some other applications than those mentioned in the introduction.

5.1 Anonymous access control

Consider a service for which it is crucial that only members of a pre-approved group can access, yet the identities of said members may be of lesser importance, or even too sensitive for the service in question. Given that one can build a profile of a user by analysing data they submit, or by combining metadata across several session, this scenario calls for anonymous authentication.

This resource acquisition may viewed even more abstractly. The resource could be access to some place or thing, such as in public transport. When a user has bought a ticket, the transport company may sign some tokens for the user that the user provides to the transportation company when entering the vehicle. This way the transport company has no ability to track the movements of individuals, but guarantee that everyone using this service has paid.

Students working for Bekk and Entur have explored this line of thought during the summer of 2021, and have used our implementation to create a proof of concept of such ticketing. To the best of our knowledge, their final report is not publicly available.

5.2 Anonymous data collection

Collection of data regarding users of different services has become big business. For a company providing a service, collecting data about how its users use this service can be a source of optimization in how they choose to develop their service. A transportation company might, based on data collected, choose to add additional buses on some routes or reduce the number of busses on other routes, or maybe create an entirely new system for their set of routes that maximizes flow and minimizes overall travel times.

With anonymous tokens, a bus company could provide everyone with tokens that last for some duration of time and track the individual within this period of time to study their travel routes. If the time duration of the token is kept short, the possibility of identifying the users could be reduced compared to longer intervals: shorter time frames means less data.

This also links in with a discussion about what should be considered personal data and what level of anonymization is acceptable.

5.3 Voting systems

With the hidden public metadata (see Section 4.1), it is possible to construct a voting system. The hidden metadata is the voting preferences for each individual voter, and the public metadata is the specific election. The signer – in this case the governing body, e.g. a government or some company – will sign a token for a voter, and the voters themselves post the ballots on a public billboard. This protocol has three phases:

Ballot stamping The voter encodes her ballot as a number b, selects random numbers t and r, and computes $t' = H(t \| b)$ and $T' = [r]H_1(t' \| \mathsf{md}_{\text{election}})$. She authenticates herself to the election authorities and sends T'. The authorities check that the voter has not voted before, and return a "stamped´´ ballot W'. The election authorities keep a record of T' and the voter's identity. The voter keeps r and t'.

Re-voting The voter proceeds as above, but submits the previous r and t' as well, which allows the authorities to list these and stamp a new ballot.

Cast ballot The voter posts the derandomized token W along with the ballot b and t. Anyone can verify that W is a valid token (i.e., properly stamped), check $t' = H(t \| b)$ against the revocation list, and then tally all b.

Notice how the revoting procedure does not leak the previous vote, as that would require the election authorities to break the preimage resistance of the hash function. The protocol also guarantees that at most one ballot is cast per person, and it provides universal verifiability.

Furthermore, the information posted on the public billboard is independent of any identities, by the properties of the token scheme.

However, the voting scheme *does* provide receipts to coercers, as a coercer may require t' from the voter, and verify that it does not appear on the revocation list.

6 Conclusion

We have demonstrated the practicality of implementing these two protocols. We have also made some modifications on the protocols such as developing batching across metadata, proved that this is safe, and further demonstrating the possible usefulness of these protocols. An additional extension to the protocols with the hidden public metadata is also demonstrated.

Our benchmarks of the protocols demonstrate the efficiency of the batch verification compared to not using batch verification. Further they demonstrate that it is are practically feasible to execure the protocols, even with a high number of tokens. The benchmarks demonstrates that NIZK protocol is somewhat more efficient than the pairing protocol, however, the pairing protocol makes up for this in not requiring the secret key for verification.

References

[1] Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. Constructing elliptic curves with prescribed embedding degrees. In Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano, editors, *Security in Communication Networks, Third International Conference, SCN 2002, Amalfi, Italy, September 11-13, 2002. Revised Papers*, volume 2576 of *Lecture Notes in Computer Science*, pages 257–267. Springer, 2002.

[2] Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda. Privacy pass: Bypassing internet challenges anonymously. *Proc. Priv. Enhancing Technol.*, 2018(3):164–180, 2018.

[3] Sharon Huang, Subodh Iyengar, Sundar Jeyaraman, Shiv Kushwah, Chen-Kuei Lee, Zutian Luo, Payman Mohassel, Ananth Raghunathan, Shaahid Shaikh, Yen-Chieh Sung, and Albert Zhang. Dit: De-identified authenticated telemetry at scale. Technical report, Facebook Inc., 2021. `https://research.fb.com/wp-content/uploads/2021/04/DIT-De-Identified-Authenticated-Telemetry-at-Scale_final.pdf`.

[4] Ben Kreuter, Tancrède Lepoint, Michele Orrù, and Mariana Raykova. Anonymous tokens with private metadata bit. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part I*, volume 12170 of *Lecture Notes in Computer Science*, pages 308–336. Springer, 2020.

[5] Henrik Walker Moe, Tjerand Silde, and Martin Strand. Anonymous tokens. `https://github.com/HenrikWM/anonymous-tokens/`.

[6] Ristretto. `https://ristretto.group/ristretto.html`. Accessed: 2021-08-11.

[7] Tjerand Silde and Martin Strand. Anonymous tokens with public metadata and applications to private contact tracing. Cryptology ePrint Archive, Report 2021/203, 2021. `https://ia.cr/2021/203`.

[8] Nirvan Tyagi, Sofía Celi, Thomas Ristenpart, Nick Sullivan, Stefano Tessaro, and Christopher A. Wood. A fast and simple partially oblivious PRF, with applications. Cryptology ePrint Archive, Report 2021/864, 2021. `https://ia.cr/2021/864`.

A Design of library

We have implemented anonymous tokens with NIZK proofs and pairings in Rust. This appendix contains a brief overview of the code available at `https://github.com/FFI-no/Paper-anonymous-tokens`. The specific protocol is abstracted by the `TokenEngine` trait[3]. This trait defines an interface to the relevant types and functions that is used in the protocols.

A.1 Interface

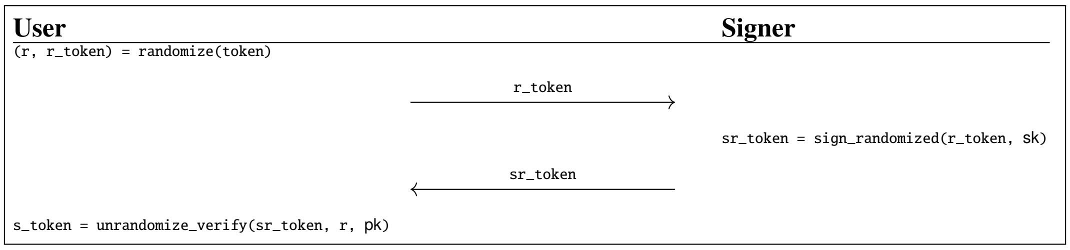
A.1.1 Generation

A token may be generated either with or without hidden public metadata. The following types and functions are provided;

```rust
type UnsignedToken;
fn generate(md: UnsignedToken::Metadata) -> UnsignedToken;
fn generate_with_hidden(md: UnsignedToken::Metadata,
                        hmd: UnsignedToken::Hidden) -> UnsignedToken;
```

A.1.2 Signing

The signing is separated into three steps, two of which the user does. The user needs to randomize the token. This randomized token is sent to the signer, and the signer may sign this token. The user may then remove the randomization from the signed randomized token, and verify that the signature is correct.



```
(r, r_token) = randomize(token)
sr_token = sign_randomized(r_token, sk)
s_token = unrandomize_verify(sr_token, r, pk)
```

There is a provided function, `sign`, that combines the steps of the user, and takes a closure that acts as an oracle for the steps that the signer does.

A.1.3 Verification

A verifier may verify that a token is signed with a certain secret key using `verify`. Depending on the protocol, the verifier either needs the public key or the private key to achieve this.

A.2 Batching

All the protocols has implemented a batched version, which is faster (see Section 4.4) for multiple tokens. This batching also implements the TokenEngine trait, and the number of tokens in a batch is a part of the type of the batch.

[3] In Rust, a *trait* describes abstract functionality that can be satisfied by implementations

B Implementation of the protocols

During the implementation of the protocols, we have not considered for side channel attacks, nor to make the implementation (consistently) constant time.

B.1 Uniform hashing

Hash functions generally output a binary string of length λ, and can be assumed to be uniform on the domain $\{0, 1\}^\lambda$. However, we sometimes model hash functions to map their input to a group G of prime order. Notice that the naïve mapping $\pi : \{0, 1\}^\lambda \to \{0, \dots p - 1\}$ defined by $n \mapsto n \mod p$ induces a skewed distribution on the image: let $2^\lambda = kp + r$ with $0 \leq r < p$, and observe that for all $n > kp$, then $\pi(n) \leq r$, which makes the lower part of $\{0, \dots p - 1\}$ somewhat more likely to be output from π. We compensate for this by resampling the hash function if $n > kp$.

For now, the security proofs assume that the output of H_m is uniform on the output group. We have unsuccessfully attempted to find an attack that exploits the any nonuniformity, as we suspect that is an artifact of the proof, and not important to security by itself.

B.2 Elliptic curves

There are two variants of the NIZK protocol implemented, one of them to be generic over any elliptic curve. The library this depends on has no interface for hashing to curves[4], so this generic implementation has to wait until this is available. Currently, a compile flag has disabled this generic implementation from use.

The other implementation of the NIZK protocol uses the elliptic curve Curve25519, and Ristretto [6] on top of this curve. The pairing based implementations uses the BLS12-381 curve [1].

[4]See issue at `https://github.com/RustCrypto/traits/issues/481`

C Examples

The `examples` in the code repository folder contains sample usage of the implementation.

C.1 Server

We have implemented a simple server using the Rocket web framework[5]. This server may act as both the signer and verifier, and is using the pairing based implementation.

The endpoints of the server:

`/keys` A GET request to `/keys/public` will return the public key in JSON format.

`/sign` A POST request to this endpoint will sign the point it is sent. The request has to contain a username, password and a token. If the user exists and is authorized for the specific resource requested, the token is signed and the signed token is sent back in JSON format. Otherwise an error is returned.

`/resource` This endpoint accepts a GET request. This request has to contain a signed token for the resource. If the token is previously unused and signed with the correct key the resource is returned. Otherwise an error is returned.

`/static` This endpoint has some static files for the website, including the QR-code webapp.

`/` Will redirect to `/static/index.html`.

The server is configured with the `Rocket.toml` file in the root of the repository, where the address of the server may be changed.

C.2 Client

The client connects to the server and gets the public key. It will try to get a token signed and use this token to access the resource twice. The first time it should succeed, and the second time fail (since the token is already used at that point).

C.3 QR Client and attacker

The QR-client connects to the server and gets the public key. It tries to get a token signed, and creates a QR-code based on this signed token. This QR-code is saved as the file `/tmp/qrcode.png`.

The QR-attacker creates its own key-pair, creates a token and signs it. A QR-code is created with this signed token and it is saved as the file `/tmp/qrcode.png`. Expected behaviour is that this token should be rejected by a correctly configured verifier.

C.4 QR-code WebApp

The webapp consists of two parts, a core written in Rust and compiled to webassembly[6], and a JavaScript wrapper around this core.

[5]Rocket web framework: `https://rocket.rs/`
[6]`https://webassembly.org/`

The wrapper prompts the user for a username, password and the ability to get a self-signed token. This is sent to the core. The generated QR-code is rendered to the screen.

If the core is asked to return a self-signed token, it will create a key- pair and a token and sign the token with this key-pair. A QR-code is generated and returned.

Otherwise, the core gets the key from the server, generates a token and tries to get the server to sign this token. A QR-code is created based on this signed token and returned.

C.5 QR-scanner box

We build a physical box for demonstration purposes. The box itself is about 8 cm $\times$ 13 cm $\times$ 21 cm. The angle of the upper panel is about 16.7°, since with this angle on a 1 m table any person of about 182 cm will have their vision orthogonal to the panel.

The internal parts are an *M5STACK Atom QR-Code*, an *Adafruit ILI9241 3.2" LCD Screen* and a *Raspberry Pi 2B 1.2*.

The QR-scanner is programmed via QR-codes[7]. This QR-scanner is connected to an *Atom lite* microcontroller. The microcontroller is programmed using the Arduino IDE, and is set up to read the information from the QR-scanner and output it as serial via USB to the Raspberry Pi.

The LCD screen is connected to the Raspberry Pi. It is connected through the SPI Interface.

The Raspberry Pi is the center of the operation. When booting up, it runs the program that connects to the server and verifies the QR-codes. If the Raspberry Pi is not connected to the network of the server, or otherwise is unable to contact the server, an error message is displayed to the screen and it will retry after some seconds.

When a network connection is established, it will get the public key of the server. It listens to the QR-scanner over serial and when a QR-code is scanned, these bytes are deserialized to a token. If this token is signed with the correct key, the server is queried to check if the token is unused. If it is valid and unused, the screen will turn green and display a message to the user. Otherwise it will turn red and an error message is displayed to the user.

The IP address of the Raspberry Pi is displayed on the top of the screen (if it is connected to a network). This may be used to SSH onto the Pi if there is trouble, e.g. to change the IP address of the server. The file `/home/pi/server_pi` should contain the IP address of the server. Update this file to let it connect to the server.

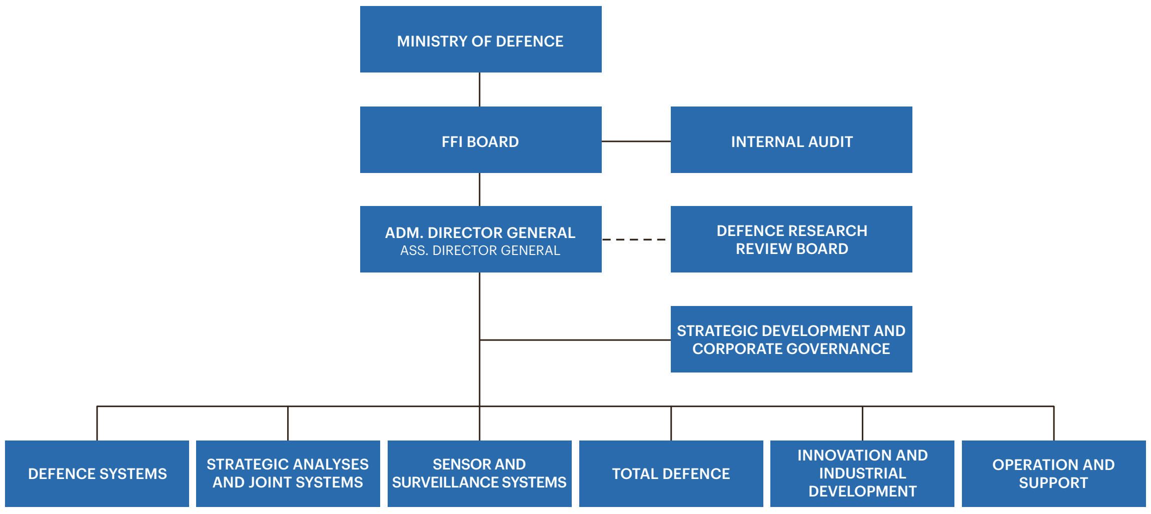
[7]M5Stack QR-code reader documentation: `https://m5stack.oss-cn-shenzhen.aliyuncs.com/resource/docs/datasheet/atombase/AtomicQR/AtomicQR_Reader_EN.pdf`

About FFI
The Norwegian Defence Research Establishment (FFI) was founded 11th of April 1946. It is organised as an administrative agency subordinate to the Ministry of Defence.

FFI's mission
FFI is the prime institution responsible for defence related research in Norway. Its principal mission is to carry out research and development to meet the requirements of the Armed Forces. FFI has the role of chief adviser to the political and military leadership. In particular, the institute shall focus on aspects of the development in science and technology that can influence our security policy or defence planning.

FFI's vision
FFI turns knowledge and ideas into an efficient defence.

FFI's characteristics
Creative, daring, broad-minded and responsible.


MINISTRY OF DEFENCE
FFI BOARD
INTERNAL AUDIT
ADM. DIRECTOR GENERAL
ASS. DIRECTOR GENERAL
DEFENCE RESEARCH REVIEW BOARD
STRATEGIC DEVELOPMENT AND CORPORATE GOVERNANCE
DEFENCE SYSTEMS
STRATEGIC ANALYSES AND JOINT SYSTEMS
SENSOR AND SURVEILLANCE SYSTEMS
TOTAL DEFENCE
INNOVATION AND INDUSTRIAL DEVELOPMENT
OPERATION AND SUPPORT