GNU Taler as a Privacy-Supporting CBDC*

March 12, 2021

GNU Taler is a token-based electronic online payment system using mathematics to secure payments, provide accountabiltiy and protect citizens' right to informational self-determination. Taler can be used by commercial banks interested in underwriting commercial e-money, or as a CBDC. In this paper, we will focus on using Taler to create a CBDC.

Taler's main contribution is that it limits the use of expensive and invasive authentication to when consumers initially withdraw funds from their commercial bank account, and to merchants when they want to receive income. When consumers spend electronic coins at a merchant, Taler only requires the consumer to authorize the transaction, but does not inherently identify the customer. By only requiring authorization to approve a payment, Taler makes payments cheaper, easier, faster and avoids easy interference with individual liberties. On the other hand, authenticating customers upon withdrawal and merchants before they can receive funds ensures that Taler still satisfies know-your-customer (KYC) and anti-money laundering (AML) regulation.

Taler is part of the GNU project, which is the original institution that created the term "Free Software", which is these days often referred to as "Free and Open Source Software". GNU Taler is released by Taler Systems SA (Luxemburg) under the GNU Public License, which grants everyone the freedom to run, copy, distribute, study, change and improve the software. This is crucial for a CBDC, as a central bank should not make itself (and by implication its economic area) dependent on proprietary software under the exclusive control of one vendor.

Taler is purely a software solution and thus does not even attempt to prevent people from cloning tokens, which in Taler are simply called coins. To prevent double-spending, Taler requires every transaction to be cleared by the bank that issued the coin. To settle a transaction, the central bank checks its database to ensure that the respective coin was not previously spent. Coins have an expiration date, allowing the database administrator to eventually discard old entries.

In the subsequent discussion of GNU Taler, we will only provide an introduction to the technology to enable readers to grasp how GNU Taler preserves privacy, how it can be integrated with the existing banking system to create a CBDC, and what the social benefits of this solution would be. Dold (2019) describes additional details, including security proofs, performance analysis and how the public key infrastructure works.

*Expanded section 4 in joint CBDC paper

1 Objectives

To build a viable CBDC, we believe the following security and privacy goals must be met:

1. Purchases must identify the buyer, and must also not be linkable to other transactions of the same buyer. Anyone who previously withdrew coins of the respective denominations from the central bank must be equally suspicious as a potential buyer for the transaction.

2. Coins must be fungible. That is, all coins signed with the same denomination key must be equivalent. In particular, must not be possible to partition the anonymity set into between users that used change and those that used cash that was directly withdrawn.

3. It must not be possible for users to hoard unlimited amounts of CBDC. The central bank must be able to impose an upper limit on the total amount of CBDC controlled by an individual. The central bank must also be able to impose daily withdrawal limits on individuals.

4. Online signing keys used by the central bank must have an expiration time to limit the impact of security breaches. Furthermore, if a key compromise is detected, the central bank must be able to revoke keys. Legitimate customers must not loose any funds or privacy in case of a key revocation. However, the attacker must not be able to deposit any counterfeit coins after keys were revoked.

5. Customers must be always able to pay any amount for which they have sufficient total CBDC, regardless of the specific denominations of the coins they may own. In other words, customers never need to be concerned with having sufficient change.

6. Anyone receiving exclusive access to funds through the system must be easily identified. It must not be possible to transfer exclusive ownership of coins via mechanisms such as giving change or dealing with expirations. However, cryptographic security is not required for this property, it is sufficient if trying to cheat income transparency is associated with substantial expected financial losses.

7. While any implementation of the system will have to make specific choices for the cryptographic primitives used, it must at least in theory be possible to replace any individual primitive used with alternative constructions. This cipher agility is useful to enable continued operation in cases where a particular primitive has been found to be potentially problematic, and also provides an upgrade path in case better constructions are invented.

Note that it is not sufficient to meet the goals most of the time, but to violate any one of the objectives under special circumstances, such as generally providing unlinkability, except not during key revocation.

2 Mathematical background

We will now describe three key building blocks for the GNU Taler protocol, including the mathematical background for *one* possible instantiation of these primitves to give an example of how an implementation could work. We note that alternative, equivalent mathematical designs exist for each of these, and we are merely presenting the simplest ones we are aware of.

2.1 Signatures

Cryptographic signatures are a key building block for many network protocols. Ordinary cryptographic signatures are used in GNU Taler to sign contracts, just like hand-written signatures for traditional contracts on paper.

While GNU Taler by default uses modern EdDSA signatures, we will present a simplistic cryptographic signature scheme based on the well-studied RSA cryptosystem here (Boneh 1999). However, we note that in principle *any* cryptographic signature scheme (DSA, ECDSA, EdDSA, RSA, etc.) can in principle be used for standard signatures in GNU Taler.

In RSA, the signer first picks two large primes p and q and computes $n = pq$ as well as Euler's Totient function $\phi(n) = (p-1)(q-1)$. Given this setup, any e with $GCD(\phi(n), e) = 1$ can be used to define a public key (e, n). The corresponding private key d is the modular inverse of $e \mod \phi(n)$, which given $\phi(n)$ can be efficiently computed using the Extended Euclidian algorithm. Euler's theorem says that $a^{\phi(n)} \equiv 1 \mod n$ for any a. A simplistic RSA signature s over m would be $s := m^d \mod n$. Verifying the signature is done by checking that $m \equiv s^e = m^{de} \mod n$. If the signature was computed correctly, the latter equation holds because $1 \equiv de \mod \phi(n)$ by our choice of d, and for any $k \in \mathbb{Z}$ Euler's theorem says that $m^{1+k\phi(n)} \equiv m \mod n$.

2.2 Blind signatures

To protect the privacy of buyers, GNU Taler uses blind signatures (Chaum 1983). The key difference between blind signatures and the vanilla signatures presented above is that with vanilla signatures, the signer learns the message m which is being signed, as is appropriate for signing contracts. In contrast, blind signatures can be used to ensure that the the signer does not learn the message at the time where the signature is being generated.

When using cryptographic signatures to create untraceable electronic cash (Chaum 1983), blind signatures can be used to prevent the central bank from tracing purchases back to the buyer. GNU Taler works in principle with any blind signature scheme, but at this time the best solution is still the traditional RSA-based variant already described by Chaum in 1983.

We will now describe the mathematics behind blind signatures using RSA. Let f be a message – such as a unique identifier for an electronic coin – to be blindly signed. Then, the receiver of the signature – in our case the customer withdrawing the coin – would first generates a blinding factor $b \in \mathbb{Z}_n$, computes its modular inverse $b^{-1} \mod n$ and transmit $f' := fb^e \mod n$ to the signer – the central bank. The signer signs f' using the usual RSA process as we described above, computing $s' := f'^d \mod n$. The receiver can then compute $s \equiv s'b^{-1} \mod n$. This works, because $f'^d = f^d b^{ed} = f^d b$ and thus multiplying

s' with b^{-1} yields f^d, which is a valid RSA signature over f as before: $s^e \equiv f^{de} \equiv f \mod n$.

2.3 Key exchange

Another central building block for secure network protocols are key exchange protocols. Here, two parties are trying to establish a shared secret starting from public information.

GNU Taler uses a key exchange protocol in an unusual way to provide a link between an original coin and change rendered for that original coin.

The most common mathematical construction for a key exchange protocol is the Diffie-Hellman construction. Here, the two parties share a generator g over a multiplicative group G of order p. Each party a has a private key $x_a \in \mathbb{Z}$ and publishes $X_a := g^{x_a} \mod p$. The security of the scheme relies on the difficulty of computing the discrete logarithm: given X_a it must be impractically hard to compute x_a. For the key exchange, party 1 receives X_2 from party 2 and computes $K := X_2^{x_1} \equiv g^{x_2 x_1} \mod p$. Similarly, party 2 receives X_1 from party 1 and computes $K := X_1^{x_2} \equiv g^{x_1 x_1} \mod p$. Both parties compute the same value K, which becomes the shared secret. The computational Diffie-Hellman (CDH) assumption states that it is difficult to compute $K = g^{x_1 x_2}$ given only g, g^{x_1} and g^{x_2}. Many groups exists where it is believed that CDH is indeed satisfied.

3 Withdrawing

We assume each customer knows a private key $x \in \mathbb{Z}_p$ and is identified with the corresponding public key $X := g^x$. The protocol is constructed such that disclosing x to another party will cause the customer to loose control over *all* of their CBDC and *all* of their privacy, providing a strong incentive for customers to keep x private. We will discuss possible means for relaxing this and the resulting implications in Section 8.

When the customer wants to withdraw a coin of some value, the customer first must obtain the associated denomination key (e, n) that the central bank is using to sign coins of that value. The central bank will typically offer denomination keys for 2^i currency units for various values of $i \in \mathbb{N}_0$, thus allowing the customer to withdraw any amount A using at most $\lceil \log_2 A \rceil$ coins.

To withdraw a coin, the customer first creates κ private transfer keys t_i for $i \in \{1, \ldots, \kappa\}$ and also computes the corresponding public keys T_i. These transfer keys are simply public-private key pairs that allow the customer to run the key exchange protocol $KX()$ κ times between x and each of the t_i. The result are three transfer secrets $K_i := KX(x, t_i)$. We note that the key exchange protocol can be used in different ways to arrive at the same value $K_i := KX(X, t_i) = KX(x, T_i)$. Given the values K_i, the customer uses a cryptographic hash function to derive values $(b_i, c_i) := H(K_i)$ where b_i is a valid blinding factor for the denomination key (e, n) and c_i is a private key suitable for both creating cryptographic signatures and also for use with the key exchange protocol. Let C_i be the public key corresponding to c_i. The customer then requests (indirectly, via the commercial bank that authenticated the customer and deducts the corresponding amount from the customer's balance) the central

bank to create a blind signature over C_i.[1] for each $i \in 1, \ldots, \kappa$. In this request, the customer also commits to the public keys T_i.

Instead of directly returning the blind signature, the central bank first challenges the customer to prove that they used the above construction correctly by providing a $\gamma \in \{1, \ldots, \kappa\}$. The customer must then reveal the t_i for $i \neq \gamma$ to the central bank. The central bank can then compute $K_i := KX(X, t_i)$ and also derive the (b_i, c_i) values. If for all $i \neq \gamma$ the provided t_i prove that customer used the construction correctly, the central bank returns the blind signature over C_γ. If the customer fails to provide a correct proof, the funds they attempted to withdraw are forfeit.

We note that someone *could* run the withdraw protocol without actually knowing x, fooling the central bank to issue them valid coins in the false belief that the customer with the public key X is the one running the withdraw protocol. To defang this problem, the central bank allows anyone who knows X to — at any time — obtain the values of T_γ and the associated blind signatures of all coins linked to account X. This allows the customer who knows x to compute $K_\gamma := KX(x, T_\gamma)$ and from there to derive (b_i, c_i) and finally to unblind the blind signature. As a result, the customer who knows x can always obtain access to all of the coins that have been withdrawn under their account. This also discourages anyone else from withdrawing coins with a value X where they do not control x, as then the person who does know x could trivially follow the link provided by the central bank and spend those coins.

4 Deposits

To pay for goods, a customer first negotiates a contract with the merchant. While the merchants cryptographically signs the contract using a vanilla cryptographic signature, the customer — their identity possibly remaining private — signs the contract with private coin keys c_i. A coin's signature on a contract with a valid coin c_i is basically an instruction from the customer to the central bank to pay the merchant who is identified by bank account in the contract. Customers may sign a contract with multiple coins if a single coin is insufficient to pay the total amount.

Furthermore, it is possible that the payment is less than the value of the coins, and that the customer is due some change. In this case, the customer not only signs the contract with the coins, but also one or more requests for change. In those requests, they use the same pattern we saw during withdrawal, except that instead of using the private key x that is tied to their identity, they use the private key c_{old} of the old coin for which they want to obtain change. The use of the key-exchange mechanism provides a link from the old coin to the change. This link ensures that whoever knows c_{old} will be able to compute the private keys of the fresh coins that are rendered as change, and thus the change will be owned by the same entity that controlled the original coin — without revealing the identity of the customer to anyone in the process.

[1] In the case RSA is used for blind signatures, we would use $f := FDH_n(C_i)$ where $FDH_n()$ is the full-domain hash over domain n

5 Refunds

The above mechanism can also be used to give anonymous customers refunds. Here, the merchant simply needs to sign a message affirming that they want to return some or all of the payment to the customer. Given such a request for a refund, the banks undo the wire transfer to the merchant and allow the customer to obtain "change" for the refunded amount.

6 Key expiration and preventing hoarding

To limit the damage a central bank could suffer from the disclosure of its private keys, a central banks would typically configure an expiration date for all signing keys. This would imply that at a set date, the coins signed by those keys become invalid. Equivalent processes exist for physical bank notes, except that physical bank notes have validity periods of decades while with electronic coins the validity period is more likely to be a few months or years.

When denomination keys expire and customers have to exchange coins signed with old denomination keys for fresh coins, the central bank could require the customer to use the withdraw protocol described above, except this time the customer would not have the amount deducted from their commercial bank account, but simply devalue their coins which are otherwise about to expire.

At this time, the central bank can easily impose a conversion limit per customer to enforce a hard limit on the amount of CBDC that any individual can hoard to this roll-over limit plus the daily withdraw limit multiplied by the validity period.

7 Revocation

Should a central bank discover that its private denomination keys may have been compromised, it must stop accepting deposits signed by those keys as such signatures might have been forged by an attacker. However, the keys are likely to have been used to issue coins to legitimate customers. Those customers must not loose their funds!

This can be achieved by having the customers reveal the blinding factors (b_γ) the customers used when withdrawing coins (or obtaining change). Given a b_gamma value, the central bank can query its database to determine whether a particular coin was legitimately (blindly) signed. If this is the case, the central bank can then refund the coin's value, either by crediting the customer's commercial bank account or by allowing the customer to withdraw fresh coins in valid denominations in response to the customer's proof of legitimate withdrawal.

We note that this does not have to impact the customer's privacy, as in the case where they withdrew a coin from their bank account, revealing the blinding factor does not relate to any transactions of the customer. Similarly, in the case that the coin was rendered as change, converting the change signed with the revoked denomination key into change singed with a valid denomination key reveals nothing about the customer.

8 Discussion

The above design creates serious repercussions for a customer who looses control over their private key x. If an attacker learns x, they can basically take control all of the CBDC owned by the customer, and if the state got hold of x, the state could – with the help of the central bank – track all of the customer's purchases. Given the limited trust society can place into law enforcement, exemplified not only by the practice of lawful seizure in the US, and the contemporary state of computer security especially for consumer electronics, this is a significant systemic risk.

It is possible to *simplify* the withdraw protocol to simply have the consumer blind a fresh coin where the private key c_i is generated at random and not derived via the $KX()$ operation from the customer's private key. We note that the other protocols (such as those for change or dealing with expiration) would continue to work with the $KX()$ as described above. With this change, a customer *could* now enable another individual to withdraw CBDC from the customer's bank account, effectively bypassing income transparency when withdrawing CBDC. This is called the withdraw loophole in Dold 2019, and the cash equivalent would be to walk to an ATM with two people, and one person authorizing a withdrawal with the other taking the cash.

While there are some scenarios where this could be abused — such as customers allowing criminals to withdraw cash from their bank accounts — the possibility to impose caps on withdraws per customer and the lack of transitivity built into the CBDC limit the social impact of this loophole. Thus, it may be beneficial to permit this loophole to avoid the associated systemic privacy risks that arise from linking all transactions of a customer to a single private key.