A provably secure keyshare protocol

Sietse Ringers David Venhoek

September 10, 2023

Abstract

For wallets holding credentials from anonymous credential schemes such as Idemix or BBS+, we introduce and prove security of a keyshare protocol that partially escrows the secret key that is shared across the wallet's credentials to a trusted third party, to allow revocation of the entire wallet in case of loss or theft as well as for increased security, enabling attaining eIDAS Level of Assurance High.

1 Introduction

In implementations of an anonymous credential scheme such as Idemix [IBM12] or BBS+ [ASM06; CDL16], one wants the user to be easily able to revoke all of their credentials in case that they lose control over the device or system storing them. In addition, securely and irretrievably storing the private key material of such credentials inside hardware is not currently supported by devices available to users, making it difficult to reach eIDAS Level of Assurance High which requires sole control of the owner over their credentials. Both of these issues can be solved by escrowing part of the attribute value(s) of the credential of a user with a trusted third party (TTP), to which the user authenticates using other (e.g. ECDSA) hardware-bound private key material. Since completing an Idemix session requires proofs of knowledge on all of the attribute values contained in the credential, such credentials are then only usable if the trusted third party is willing to cooperate with the user of the credential. This allows revocation of all of a users credentials by simply instructing the trusted third party to no longer cooperate on proofs of knowledge of its attribute value(s).

In IRMA[1] [22a; 22b], an implementation of the Idemix anonymous credential scheme, this is implemented through splitting the value of the 0-th attribute (the user secret key) into two parts: m_u, which is kept by the user of the credential, and m_t, which is held by the trusted third party. The value of the 0-th attribute as a whole is then taken to be $m = m_u + m_t$.

[1] To make things specific, in the remainder of this paper we work in the context on Idemix. Since the security proof below depends only on the hardness of the discrete logarithm problem, however, nothing prevents one from implementing the protocol in this paper on BBS+ or other (sufficiently similar) anonymous credential schemes. In such cases care should be taken to make the appropriate corrections in the protocol; for example, in the case of BBS+ as well as any (elliptic curve-based) system in which the group order is public knowledge, all responses $s = w + cm$ of zero knowledge proofs must be taken modulo the group order, while this is in fact impossible in Idemix (in which the group order is the issuer's private key).

User		Verifier
N, m, R		$N, P = R^m \mod N$
		Choose random η
	$\xleftarrow{\eta}$	
Choose random w		
$W = R^w \mod N$		
$c = \mathcal{H}(\eta, P, W)$		
$s = cm + w$		
	$\xrightarrow{c,s}$	
		$W = R^s P^{-c} \mod N$
		Verify $c = \mathcal{H}(\eta, P, W)$

Figure 1: Non-interactive proof of knowledge of a discrete logarithm.

During issuance or disclosure of attributes, a proof of knowledge on the combined attribute m can then be constructed by having the user of the credential construct a Schnorr zero-knowledge proof [Sch90] of m_u, and the trusted third party construct a Schnorr proof of m_t. These proofs can then be merged by multiplying the commitments, and adding the responses, yielding a Schnorr proof for the combined secret m. This protocol is illustrated in Figure 2. Note that it is in essence two runs of the Schnorr protocol, one between the trusted third party and the user, and one between the user and the verifier. The only difference is that the user uses the output of the trusted third party to construct the proof that the verifier is asking for, instead of fully providing it himself.

Since the user at no point knows m as a whole, nor m_t specifically, and since all credentials of the user share the same value m as the 0-th attribute, the above is sufficient to be able to block all credentials of a user, as long as we can guarantee that the trusted third party was actually used during issuance of the credential.

This paper focuses on how to achieve this. Specifically, we will define and prove security of a protocol that allow the issuer or verifier to know that without collaboration between user and trusted third party, the user cannot control or know the value m during or after credential issuance.

As we will be working within the context of Idemix, by convention all products and exponentiations will henceforth be taken modulo $N = pq$.

TTP		User		Verifier
$N, R; m_t$		$N, R; m_u, P_t$		N, R
				choose random nonce η
			$\xleftarrow{\eta}$	
		choose random w_u $W_u = R^{w_u} \mod N$ $h_W = \mathcal{H}(P, W_u)$		
	$\xleftarrow{h_W}$			
choose random w_t $W_t = R^{w_t} \mod N$				
	$\xrightarrow{W_t}$			
		$W = W_u W_t$ $c = \mathcal{H}(\eta, P, W)$ $s_u = w_u + c m_u$		
	$\xleftarrow{\eta, s_u, P, W_u}$			
verify $h_W = \mathcal{H}(P, W_u)$ $c = \mathcal{H}(\eta, P, W_u W_t)$ $s_t = w_t + c m_t$ $s = s_u + s_t$ $\sigma = \text{sign}(\text{sk}, (c, s))$				
	$\xrightarrow{\sigma, s}$			
			$\xrightarrow{\sigma, c, s, P}$	
				verify $(\text{pk}, \sigma, (c, s))$ $W = R^s P^{-c} \mod N$ verify $c = \mathcal{H}(\eta, P, W)$

Figure 2: The keyshare protocol.

2 The protocol

We start with the simplest case of proving knowledge of a single exponent. Let $m = m_u + m_t$, in which m_t is known by the trusted third party (TTP) but not by the user, and m_u is known by the user but not by the TTP. Let R be some publicly known value (e.g. part of a public key), and let $P_t = R^{m_t}$ be known to the TTP and $P_u = R^{m_u}$ be known to the user. Then the protocol in Figure 2 allows the user and the TTP to jointly prove knowledge to some verifier[2] of the number m satisfying $P = P_u P_t \mod N = R^m \mod N$, without disclosing m, and without any party knowing the value of m. This protocol is designed as follows.

- The user must compute both of its contributions to the zero-knowledge proof of m before it gets access to those of the TTP:
  - In the first step this is realized by the TTP requiring the user to send $h_W = \mathcal{H}(P, W_u)$ before it will respond with its commitment W_t. The user does not send its commitment W_u directly in this step to prevent the TTP from being able to let its choice of W_t depend on W_u. When the user later sends s_u, it also sends P and W_u so that then the TTP can verify $h_W = \mathcal{H}(P, W_u)$. This forces the user to use the same W_u in the entire protocol, just like it would be if it had sent W_u directly.
  - The user must send its response s_u to the TTP before it gets to see the response s_t of the TTP. The TTP additionally signs the final response $s = s_u + s_t$, so the user cannot lie to the verifier which s_u it used.
- The TTP enforces that the challenge is constructed correctly according to the Fiat-Shamir heuristic.
- The verifier ultimately receives a conventional Schnorr zero-knowledge proof in the Fiat-Shamir heuristic [FS87] of the secret $m = m_u + m_t$, along with a signature σ from the TTP over s and c. What the verifier receives is thus very close to conventional Idemix.

We adopt the following notational convention for the variables P, W, m, s occuring in a zero-knowledge proof.

- The subscript u on a variable, such as P_u, denotes that this value belongs to the user;
- Similarly, the subscript t on a variable, such as P_t, denotes that this value belongs to the TTP;
- When a variable that can occur with the u or t subscripts occurs without a subscript, e.g. P, this means that it concerns the sum or product of the subscripted variables. For example, $P = P_u P_t$ and $s = s_u + s_t$.

2.1 Security games

We adopt the following notations.

[2] In an anonymous credential scheme this protocol is most useful during issuance, since if the collaboration of the TTP is enforced during issuance of a credential then the collaboration of the TTP is automatically guaranteed for every future disclosure involving that credential. In terms of terminology, however, in the remainder of this section we stick with "verifier" instead of "issuer", because in the context of this section the only role of that party is to verify the proof of knowledge of $P = R^m \mod N$. We return to issuance in Section 3.1.

- We adopt the random oracle model (ROM), modelling a cryptographic hash function as a random oracle $\mathcal{H}$. That is, $\mathcal{H}$ assigns a random integer to every input. If $\mathcal{H}$ is called with an input already specified previously, it consistently returns the same integer as returned previously. In security games with a challenger and an adversary, this oracle is controlled by the challenger.
- $\mathcal{C}(i, h_W)$ encodes the TTP first interaction, where it receives a value h_W and some arbitrary label i and returns a commitment W_t. This function terminates if called multiple times with the same value of i.
- $\mathcal{R}(i, \eta, s_u, P, W_u)$ encodes the second TTP interaction: when given η, s_u, P, W_u and a label i, it returns a pair (σ, s) such that (1) σ is a signature over (c, s) where $c = \mathcal{H}(\eta, P, W_u W_t)$ and W_t is the value in the corresponding call to $\mathcal{C}(i, h_W)$, and (2) for $s_t = s - s_u$, it holds that $R^{s_t} = W_t P_t^c$. This function terminates if called multiple times for the same i, or called for a value of i for which there is no matching invocation of $\mathcal{C}(i, h_W)$, or if $h_W \neq \mathcal{H}(P, W_u)$.

The label i keeps track of which call to $\mathcal{R}$ belongs to which call to $\mathcal{C}$. A call to $\mathcal{C}$ and $\mathcal{R}$ with the same label i can be said to constitute a TTP session. In this paper, between the two invocations of $\mathcal{C}(i, \cdot)$ and $\mathcal{R}(i, \cdot)$ for a particular TTP session i, the user is allowed to call $\mathcal{C}$ and $\mathcal{R}$ with other session labels as it sees fit. We use this to prove security in the presence of adversaries controlling multiple distinct users with which they perform multiple interleaved keyshare sessions simultaneously.

We now formalize the requirements on the user stated in the previous section. We do this in the form of a security game, for which we will show that an adversary needs to violate the hardness of the Discrete Logarithm problem in order to win.

Game 1. *The adversary participates in Protocol 2 with the TTP and verifier in the role of the user, having access to functions $\mathcal{C}(i, h_W)$ and $\mathcal{R}(i, \eta, s_u, P, W_u)$ of the TTP, which it may invoke any number of times in any order. After finishing this, its access to the TTP is removed. It then participates in Protocol 1 with the challenger, in the role of user. It wins if it makes the challenger accept.*

The above game is somewhat unwieldly to directly build security proofs on. Since it ends with an execution of Protocol 1 which is a conventional Schnorr zero-knowledge proof, we can however extract its secret from it in the usual fashion if it wins the game. This results in the following equivalent but simpler game.

Game 2. *The adversary participates in Protocol 2 with the TTP and verifier in the role of the user, having access to functions $\mathcal{C}(i, h_W)$ and $\mathcal{R}(i, \eta, s_u, P, W_u)$ of the TTP, which it may invoke any number of times in any order. It wins if it outputs a number a such that $P = R^a \mod N$.*

2.2 Security proof

Theorem 1. *Assuming the hardness of the discrete logarithm problem of integers modulo N, no probabilistic polynomial-time algorithm can win Game 2 with non-negligible probability.*

Proof. In the security proof of the ordinary Schnorr proof of knowledge protocol, the fact that the challenge is under control of the challenger plays a crucial role. This allows it to run the adversary twice with different challenges c and c', resulting in an efficiently computable formula for the secret, contradicting the hardness of the discrete logarithm problem. This holds even in the Fiat-Shamir heuristic, where the challenge is still under the control of the adversary through the random oracle $\mathcal{H}$, whose output it may freely choose.

This proof is structured similarly. The protocol is such that the adverary is forced by the verifier and TTP to use an honestly constructed challenge $c = \mathcal{H}(\eta, P, W_u W_t)$, again giving control over the challenge to the challenger (which controls the verifier and TTP). Together with some probabilistic bookkeeping, this allows the proof to proceed.

Assume that an adversary exists that can win Game 2 with non-negligible probability. We use the adversary to construct an algorithm that when given some number P_t can compute $\log_R(P_t) \mod N$ in polynomial time. Acting as the challenger which controls the TTP and verifier, we play Game 2 with the adversary, which acts as the user, in the following (standard) way.

If the adversary receives protocol messages $(x_1, \ldots, x_n)$ denote its output with $\mathcal{A}_{r,p,w}(x_1, \ldots, x_n)$, where r is the randomness source used by $\mathcal{A}$, and where p and w are the public and private input to the adversary, respectively. The function $\mathcal{A}_{r,p,w}$ is called the next-message function. Since this function exists since we assumed the adversary to exist, we can make the challenger invoke $\mathcal{A}_{r,p,w}(x_1, \ldots, x_n)$ at will (controling the r and p and the x_i but not controlling or knowing w). The challenger can use this next-message function to perform the protocol with the adversary, with a high degree of control: it can "pause" the adversary by not invoking the next function, and even "rewind" the adversary to an earlier phase of the protocol, by removing or changing the last parameter(s) x_n, and then proceed again with other parameters to its advantage.

The adversary may call $\mathcal{C}$ and $\mathcal{R}$ as well as the random oracle $\mathcal{H}$ at any time with any input values. When it does, the challenger acts as follows.

- $\mathcal{C}(i, h_W)$: Choose random integers c and s_t, and compute the TTPs commitment as $W_t = R^{s_t}/P_t^c \mod N$. Out of all invocations so far to $\mathcal{H}$, see if there is one receiving two arguments where h_W was returned. If such an invocation exists, i.e. $h_W = \mathcal{H}(P, W_u)$ for some P and W_u supplied by the user, take P and W_u and modify $\mathcal{H}$ such that future calls to $\mathcal{H}(\eta, P, W_u W_t)$ return the number c. Return W_t.
- $\mathcal{R}(i, \eta, s_u, P, W_u)$: Lookup the invocation to $\mathcal{C}$ with the same label i. If no such invocation of $\mathcal{C}$ exists, then abort. Perform the steps as defined in the protocol, but instead of honestly computing s_t (which we can't do because we don't know m_t), use the s_t chosen during the computation of $\mathcal{C}(i, h_W)$.

The challenger uses these functions to answer TTP queries of the adversary as it runs, and lets it invoke them as it sees fit, until finally it either produces an output or aborts.

Since we assumed that the adversary can win Game 2 which defines precisely how $\mathcal{C}$, $\mathcal{H}$, $\mathcal{R}$ must act, we must show that these functions behave indistinguishably from the $\mathcal{C}$, $\mathcal{H}$, $\mathcal{R}$ of an honest TTP:

- $\mathcal{C}$ always returns randomly distributed numbers, just as it would for an honest TTP.
- The same holds for $\mathcal{H}$.

- As to $\mathcal{R}$, given its input first it computes $\mathcal{H}(P, W_u)$ and checks that that equals h_W. If this does not hold, then like an honest TTP, $\mathcal{R}$ aborts. If it does hold, then (P, W_u) must equal the preimage of $h_W = \mathcal{H}(P, W_u)$ found by $\mathcal{C}$. Looking at how $\mathcal{C}$ programmed $\mathcal{H}$, we see that when $\mathcal{R}$ computes $\mathcal{H}(\eta, P, W_u W_t)$ it will receive the number c such that $R^{s_t} = P_t^c W_t$ as required. Like an honest TTP, therefore, $\mathcal{R}$ returns a tuple (σ, s), where s is a randomly distributed number satisfying the expected relation, and where σ is a signature over (c, s).

Next, note that since the signature σ is unforgable, the output of the adversary always has to correspond with the result from one of the TTP sessions it has run, i.e. $\mathcal{R}(i, \eta, s_u, P, W_u)$ for some i. This implies that we don't have to worry about the case where the adversary provides values for its output without invoking $\mathcal{R}$. Using the queries and output of our adversary, our goal will now be to extract two traces of the adversary, where at the end the adversary uses the same session i in the results it provides, but with different challenges. If we manage to get that, then the regular approach for extracting a secret from Schnorr proofs can be applied, after which we can use the secret to compute $\log_R(P_t)$.

First we run the adversary once to completion. If it aborts, we abort. Suppose it succeeds. Then we have obtained from the adversary the values P, W_u, s_u, c as well as a, and we know the following:

1. $P = R^a$, by the assumption that the adversary succeeds.
2. $W = R^s P^{-c}$, by the verification done by the verifier.
3. In its message to the verifier, the user is forced by the signature σ to send the challenge $c = \mathcal{H}(\eta, P, W_u W_t)$ as constructed by the TTP in the invocation of $\mathcal{R}$. Additionally, for this c the verifier verifies $c = \mathcal{H}(\eta, P, W)$, which is only going to hold if the verifier and TTP use the exact same input parameters to $\mathcal{H}$. Therefore, $W = W_u W_t$.
4. The user is forced by the signature σ to use $s = s_u + s_t$ as the response for the proof of knowledge of P.
5. $W_t = R^{s_t} P_t^{-c}$, by construction of $\mathcal{C}$ and $\mathcal{R}$.

Using each of these points sequentially, we first compute

$$R^{ac} = P^c = \frac{R^s}{W} = \frac{R^s}{W_u W_t} = \frac{R^{s_u + s_t}}{W_u R^{s_t} P_t^{-c}} = \frac{R^{s_u} P_t^c}{W_u}$$

Rewriting the left and right hand sides of this, we find the following intermediate expression for P_t^c:

$$P_t^c = W_u R^{ac - s_u} \tag{1}$$

Using c, we now look up the associated TTP session i in our log of TTP queries performed by the adversary. Let us rewind the adversary to the point of the call to $\mathcal{C}$ in TTP session i. Now, modify $\mathcal{H}$ so that the challenge for that TTP session becomes a new random value c'. This allows us to then run the adversary until it again makes the $\mathcal{R}$ call with the label i that it used previously when it succeeded. Pause the adversary at the point of this call (note that we could not have continued here if it were necessary, as we can now produce no value s' that would make the verifier accept if the adversary were honest). Let s'_u be the third argument to that invocation of $\mathcal{R}$. If it does not call $\mathcal{R}$ again with the label i, then we abort.

Suppose that the adversary indeed calls $\mathcal{R}$ with label i. At this point, there are two possible scenarios. The following equation either holds, or it does not:

$$P_t^{c'} = W_u R^{ac' - s'_u} \tag{2}$$

That is, equation (1) for c' and s'_u. Let us for now assume that it does hold (otherwise we simply abort). Since $c \neq c'$, we may combine both expressions to obtain one without W_u in it:

$$P_t^{c-c'} = R^{a(c-c')-(s_u - s'_u)}$$

which finally results in

$$P_t = R^r \quad \text{with} \quad r = a - \frac{s_u - s'_u}{c - c'}.$$

As we can calculate r in polynomial time from the information we have gathered, and as all our steps running the adversary also only take polynomial time, we have now provided a method for calculating discrete logarithms in polynomial time. It remains to show that this method succeeds with non-negligible probability.

The probability that this method succeeds equals the probability that it does not abort at each of the places above where it aborts under certain circumstances. Denote the probability that the method described above to compute r works with Pr[success], and denote the adversary with $\mathcal{A}$. Additionally, we write $\Pr_1$ or $\Pr_2$ to refer to a probability occuring in the first or second run of the adversary. Then we can summarize our observations so far schematically as follows:

$$\begin{aligned} \Pr[\text{success}] = \; & \Pr_1[\mathcal{A} \text{ wins}] \\ & \times \Pr_2[\mathcal{A} \text{ invokes } \mathcal{R}(i,\cdot)] \\ & \times \Pr_2[\text{Equation (2) holds} \mid \mathcal{A} \text{ invokes } \mathcal{R}(i,\cdot)] \end{aligned}$$

The first factor is non-negligible by assumption. What remains to show is that the second and third factors are also non-negligible. To see this, observe that as we pause the adversary in the second run, from its perspective it cannot detect that it is running in its first or second run, or indeed that it is being used as described above, as opposed to just running normally. We repeatedly use this observation in each of the following points:

- The label i is important to us since the adversary used that label in its response during the first run, but from its perspective, there is nothing special about label i distinguishing it from any of the others. This means that

$$\begin{aligned} & \Pr_2\left[\text{Equation (2) holds} \mid \mathcal{A} \text{ invokes } \mathcal{R}(i,\cdot)\right] \\ & \qquad = \Pr_2\left[\text{Equation (2) holds}\right]. \end{aligned}$$

- If instead of pausing the adversary when it calls $\mathcal{R}(i,\cdot)$, we somehow could provide it with the suitable value for s', then Equation (2) would have to hold for the adversary to be succesful. Therefore, the probability that this holds is at least as big as the probability that the adversary would win the second time:

$$\Pr_2\left[\text{Equation (2) holds}\right] \geq \Pr_2\left[\mathcal{A} \text{ wins}\right].$$

- Going further, we must have $\Pr_1[\mathcal{A}\text{ wins}] = \Pr_2[\mathcal{A}\text{ wins}]$. Therefore, we can just write $\Pr[\mathcal{A}\text{ wins}]$.

Gathering our remarks so far, we now have

$$\Pr\left[\text{success}\right] \geq \Pr\left[\mathcal{A}\text{ wins}\right]^2 \times \Pr_2\left[\mathcal{A}\text{ invokes }\mathcal{R}(i,\cdot)\right].$$

As to the second factor in this inequality, our observation above allows us to treat the adversary's choice of which TTP session it uses for its output as a discrete stochastic variable X taking integer values in a range $[1, M]$, where M is the maximum possible numer of TTP sessions the adversary would ever make. Now, from Lemma 1, it follows that the probability of two independent random trials on X giving the same outcome is at least $1/M$. Therefore, we finally obtain

$$\Pr\left[\text{success}\right] \geq \Pr[\mathcal{A}\text{ wins}]^2 \times \frac{1}{M}.$$

Since the adversary is polynomial time in the security parameters, it can do at most a polynomial number of TTP sessions. Therefore, $1/M$ is non-negligible. Since $\Pr[\mathcal{A}\text{ wins}]$ is non-negligible by assumption, this completes the proof. □

3 Implementation

The above protocol works for a zero-knowledge proof of a single exponent m. In practice, a disclosure proof in Idemix and IRMA always involves zero-knowledge proofs of more than one exponent simultaneously. In the next subsections, we will generalize the protocol above to the following:

1. Issuance of a single credential;
2. Disclosure of attributes from a single credential;
3. Issuance of multiple credentials simultaneously with disclosure using multiple credentials;
4. Issuance of multiple credentials simultaneously with disclosure using multiple credentials, possibly involving more than one TTP simultaneously.

In the remainder we now use R_0 for what was previously called R, but the rest of the notation from the previous section we keep as it was.

3.1 Issuance

During issuance, the user proves knowledge of $U = S^v R_0^{m_u} P_t \prod_{i\in B} R_i^{m_i'}$, where B is the set of indices of randomblind attributes, and m_i' is the user's contribution to those attributes.

Above, the challenge c was constructed in the Fiat-Shamir heuristic as $c = \mathcal{H}(\eta, P, W)$. In the IRMA implementation of Idemix, the challenge actually looks as follows:

$$c = \mathcal{H}(\texttt{context},\ U,\ W,\ \ \eta) \tag{3}$$

where:

- `context` is a number always equal to 1.
- $U = S^v R_0^{m_u} P_t \prod_{i\in B} R_i^{m_i'}$ with B and m_i' defined as above.

- $W = W_u W_t$ with $W_u = S^{w_v} R_0^{w_u} \prod_{i \in B} R_i^{w_i}$, where w_v and w_i are the randomizers for the zero-knowledge proofs over v and m'_i.

To take this into account, we change the following.

- The win condition for the adversary of Game 2 becomes as follows: the adversary must output (v, a) such that $U = S^v R_0^a$.
- Throughout the protocol, the user, TTP and issuer use the value U instead of P, and construct the challenge c as in Equation (3).

Notice that contrary to the protocol as defined in Section 2, the second point means that here the TTP no longer has the ability to learn P_u, since P_u is information-theoretically hidden in the ephemeral value $U = S^v P_u P_t$ through the random number v.

Security proof

The security proof in the previous section relied on the fact that after execution of $\mathcal{C}(i, h_W)$, the challenger (which controls the TTP, issuer as well as the oracle $\mathcal{H}$) knows all input parameters to $\mathcal{H}$ that an honest user would use when constructing the challenge c. The extension here is constructed such that it has the same property, due to how the extra parameters to $h_W = \mathcal{H}(\cdot)$ as well as those to $c = \mathcal{H}(\cdot)$ are chosen. Indeed, if

$$\mathcal{C}(i, \mathcal{H}(U, W_u)) = W_t,$$

then the correct challenge would be

$$c = \mathcal{H}(\texttt{context},\ U,\ W_u W_t,\ \eta),$$

all of whose arguments are known to the challenger after execution of $\mathcal{C}$. Therefore, in the security proofs we do not need to change the definitions of $\mathcal{C}$, $\mathcal{H}$, $\mathcal{R}$ as used by the challenger, apart from taking into account the changed input parameters.

Next, we obtain an expression for P_t^c like Equation (1) as follows. In the remainder of this subsection, we assume for simplicity that no randomblind attributes are being issued ($B = \emptyset$). This makes no difference to the security proof. After the first run of the adversary, we have values satisfying the following.

1. $U = S^v R_0^a$, by the assumption that the adversary succeeds.
2. $W = S^{s_v} R_0^s U^{-c}$, by the verification done by the issuer.
3. In its message to the issuer, the user is forced by the signature σ to send the challenge $c = \mathcal{H}(\eta, U, W_u W_t)$ as constructed by the TTP in the invocation of $\mathcal{R}$. Additionally, for this c the issuer verifies $c = \mathcal{H}(\eta, U, W)$, which is only going to hold if the issuer and TTP use the exact same input parameters to $\mathcal{H}$. Therefore, $W = W_u W_t$.
4. The user is forced to use $s = s_u + s_t$ as the response for the proof of knowledge of P.
5. $W_t = R_0^{s_t} P_t^{-c}$, by construction of the TTP.

Using these sequentially as before, we compute

$$S^{cv} R_0^{ca} = U^c = \frac{S^{s_v} R_0^s}{W} = \frac{S^{s_v} R_0^s}{W_u W_t} = \frac{S^{s_v} R_0^{s_u + s_t}}{W_u R_0^{s_t} P_t^{-c}} = \frac{S^{s_v} R_0^{s_u} P_t^c}{W_u}$$

which results in

$$P_t^c = W_u S^{cv-s_v} R_0^{ca-s_u}.$$

After the second run of the adversary, we obtain the same expression but with c', s'_v and s'_u. Combining those as we did previously, this results in the expression

$$P_t = S^{r_v} R_0^r, \quad \text{where} \quad r_v = v - \frac{s_v - s'_v}{c - c'}, \quad r = a - \frac{s_u - s'_u}{c - c'}.$$

The ability to compute such x and y is equivalent to the ability to compute discrete logarithms. To see this, suppose that before interacting with the adversary, the challenger constructs the value S as $S = R_0^s$ for some random number s. This does not change the behaviour of the challenger towards the adversary in any way, so the adversary cannot tell that the challenger has such knowledge of the exponent s. Therefore, this makes no difference to the security proof. Then the expression above becomes

$$P_t = S^{r_v} R_0^r = R_0^{sr_v + r}.$$

The remainder of the proof stays the same.

3.2 Disclosure

For ease of notation, suppose the user wishes to disclose none of the attributes in her credential, i.e. hide all of them in the zero-knowledge proof. Then the user proves knowledge of

$$Z = A^e S^v R_0^{m_u} P_t \prod_i R_i^{m_i},$$

Additionally, the Fiat-Shamir challenge is constructed as

$$c = \mathcal{H}(\texttt{context}, A, W_u W_t, \eta). \tag{4}$$

with the user commitment $W_u = A^{w_e} S^{w_v} R_0^{w_u} \prod_i R_i^{w_i}$. Note that the hash inputs differs a little from what we have seen in previous sections: there, the second and third input parameters to $\mathcal{H}$ were always computed using the same formula, once with the secret(s) and once with the randomizer(s) as the exponents. Instead, here the second parameter is A. However, the TTP does nothing with this parameter except enforce correctness of h_W and putting it into $\mathcal{H}$ when computing the challenge, just as it previously did for U and P. Therefore, this makes no difference for the rest of the argument.

We make the following changes. In the remainder of this paper, we will call the party with which the user is performing disclosure or issuance the requestor. Additionally:

- The win condition for the adversary of Game 2 becomes as follows: the adversary must output $(a, e, v, m_1, \dots, m_k)$ such that $Z = A^e S^v R_0^a \prod_i R_i^{m_i}$.
- Throughout the protocol, the user, TTP and requestor use the value A instead of P, and construct the challenge c as in Equation (4).

Using the same reasoning as above, this will result in an expression of the form

$$P_t = A^{r_e} S^{r_v} R_0^r \prod_i R_i^{r_i}.$$

As before, the ability to compute the exponents in such an expression is equivalent to the ability to compute discrete logarithms.

3.3 Disclosure and/or issuance of multiple credentials

In practice, the keyshare protocol should allow users to perform a single session in which multiple credentials are issued simultaneously (zero or more), combined with the disclosure of attributes out of multiple credentials (zero or more). Henceforth, we label each U and A and W with an index counting the involved credential. Denote the number of credentials being issued (as opposed to disclosed) with $k-1$. For such a session, the challenge would look as follows:

$$\begin{aligned} c = \mathcal{H}(&\texttt{context}, \\ &U_1, W_{1,u}W_t,\ U_2, W_{2,u}W_t,\ \ldots, \\ &A_k, W_{k,u}W_t,\ A_{k+1}, W_{k+1,u}W_t,\ \ldots, \\ &\eta). \end{aligned} \tag{5}$$

That is, for each credential involved two parameters are put into $\mathcal{H}$: once the number being proved knowledge of (U_i or A_i) and once the corresponding commitment. The user's contributions to these commitments (i.e., $W_{i,u}$ etc.) differ for each of the credentials involved, but the TTP's contribution W_t is the same each time, because the TTP's contribution to the proof of knowledge is always over the same number m_t.

The number h_W must now be computed as follows:

$$h_W = \mathcal{H}(U_1, W_{1,u},\ U_2, W_{2,u},\ \ldots,\ A_k, W_{k,u},\ A_{k+1}, W_{k+1,u},\ \ldots)$$

$\mathcal{R}$ now receives the following parameters:

$$\mathcal{R}(i, \eta, s_u,\ U_1, W_{1,u},\ U_2, W_{2,u},\ \ldots,\ A_k, W_{k,u},\ A_{k+1}, W_{k+1,u},\ \ldots)$$

Since the TTP must compute a multiplication for each commitment modulo the modulus of the public key, it needs to know which public keys are involved. Therefore, whenever a value A_i or U_i or W_i is sent or used as input to $\mathcal{H}$, we assume that an identifier of the issuer public key is included. We will for legibility however not include this in our notations.

Using these input parameters, $\mathcal{R}$ checks that h_W was correctly computed. Next, it computes the challenge as in Equation (5), and then proceeds with the protocol normally.

Using any one of the involved credentials, the argument of one of the two preceding subsections may then be used to solve $\log_{R_0}(P_t)$.

3.4 Using another TTP or no TTP for some credentials

Finally, we want the protocol to allow the user to use other TTPs, or no TTP at all, for some credentials involved in the session. For example, IRMA's main production scheme does use a TTP while its demo scheme does not, and currently IRMA supports issuance and disclosure sessions in which some credentials are from the production scheme while others are from the demo scheme.[3] The protocol developed here should not make that impossible.

[3]Normally, when all credentials involve the same TTP (or when all of them use no TTP at all), the secret (the zeroth attribute) is forced to have the same value by the requestor, in order to prevent credential pooling attacks. When not all credentials use the same TTP, then the secrets will have different values $m_u + m_t$ and $m_u + m'_t$. These values will not be equal, and so in such cases the requestor will not require them to be equal.

We achieve that as follows. We require that an issuer only ever uses a single TTP for issuance of all of its credentials, and we assume that all participants know which issuers use which TTPs (in IRMA, this is achieved using IRMA schemes). In addition, for ease of notation, if the user uses no TTP for a credential, then we assume that it uses a fictional TTP for that credential that uses $m_t = 0$ and $W_t = 1$.

The challenge c must now be constructed as follows.

$$\begin{aligned} c = \mathcal{H}(&\texttt{context}, \\ &U_1, W_1,\ U_2, W_2,\ \ldots, \\ &A_k, W_k,\ A_k, W_{k+1},\ \ldots, \\ &\eta), \end{aligned} \tag{6}$$

where now for each of the W_i, one of the following must be the case:

1. If credential i uses this TTP, then in the challenge c as constructed by the user and the TTP, the value W_i must be of the form $W_i = W_{i,u}W_t$ as before;
2. If not, then the user has received a W'_t from another TTP. It constructs $W_i = W_{i,u}W'_t$ and sends that to the current TTP, who uses W_i as is (that is, without multiplying it with its own W_t) in the computation of c.

As mentioned before, in order to keep the security proof working, by the time $\mathcal{C}(i, h_W)$ is invoked the challenger must be able to construct the challenge c that an honest user would use. Since for some i the TTP must include in the commitments its contribution W_t during the computation of the challenge $c = \mathcal{H}(\cdot)$, while for others it must not, we now require that the user indicates so for each credential in its computation of $h_W = \mathcal{H}(\cdot)$. Since the challenger controls the hash function $\mathcal{H}$, this provides it with the require information. The value h_W must therefore now be computed as follows:

$$\begin{aligned} h_W = \mathcal{H}(&b_1, U_1, W_{1,u},\ b_2, U_2, W_{2,u}, \ldots \\ &b_k, A_k, W_{k,u},\ b_{k+1}, A_{k+1}, W_{k+1,u},\ \ldots) \end{aligned} \tag{7}$$

Here, b_i is a bit indicating whether or not the TTP must include its contribution W_t to the commitment $W_{i,u}$.

For notational ease we assume the simplest possible case, of two TTPs are involved in one session; the generalization to more TTPs will be obvious. When invoking $\mathcal{C}$ at each TTP, the user must send to them the following values for h_W:

$$\begin{aligned} &\mathcal{H}(1, U_1, W_{1,u},\ 0, A_2, W_{2,u}), \\ &\mathcal{H}(0, U_1, W_{1,u},\ 1, A_2, W_{2,u}). \end{aligned} \tag{8}$$

Denote the commitments of the TTPs with which they respond with W_t and W'_t. Then if the protocol is to succeed, the user and both TTPs must be able to construct the following challenge:

$$c = \mathcal{H}(\texttt{context},\ U_1, W_{1,u}W_t,\ A_2, W_{2,u}W'_t,\ \eta).$$

In the security proof we may assume that all participating TTPs are controlled by the challenger. Therefore, by the time $\mathcal{C}(i, h_W)$ is invoked the challenger knows both W_t and W'_t, as well as all arguments passed in the expressions 8. This indeed suffices to be able to compute this challenge.

In implementations, the first TTP knows W_t and can compute $W_{1,u}W_t$ normally, thereby enforcing as it should that its contribution W_t to the commitment is taken into account. However, as can be seen from the expression for c above, it must also know $W_{2,u}W'_t$ involving the W'_t of the second TTP. The user therefore has to send the product $W_{2,u}W'_t$ to the first TTP, and similarly $W_{1,u}W_t$ to the second TTP, in the invocations of $\mathcal{R}$ at each TTP, so that the TTPs can use those products as is in the construction of the challenge c.

When interacting with a TTP, for each credential this construction gives to the user the choice to include the TTP's contributions to the zero knowledge proof of the secret m, or not. However, the requestor who finally verifies the proof knows the issuer of each involved credential, so it also knows which TTP should be involved for those credentials. It can therefore force the user to use the correct TTP for each credential, by requiring a valid signature σ from the appropriate TTP for each credential. Thus, for each credential the user is still required to use the appropriate TTP.

Summarizing, this construction allows the user to use multiple TTPs in a single session as follows.

1. First, it computes all commitments $W_{i,u}$ as before.
2. Next, for each TTP it computes a value h_W using Equation (7), setting b_i to 1 for the credentials for which it wants to use the TTP.
3. It invokes $\mathcal{C}(i, h_W)$ of each TTP using the values h_W computed earlier. From each TTP, it receives a value W_t in response.
4. For each credential i it computes $W_i = W_{i,u}W_t$, where W_t is the value it received from the TTP that it uses for credential i, and then it computes the challenge c using Equation (6).
5. It invokes $\mathcal{R}$ for each TTP, sending to each TTP not only what it normally does but also the product $W_i = W_{i,u}W_t$ for each proof not involving that TTP. When computing the challenge c, the TTP constructs $W_i = W_{i,u}W_t$ itself if the corresponding bit in h_W was set to 1 and otherwise uses the product $W_{i,u}W_t$ that it received from the user. From each TTP the user receives a signature σ and a response s such that σ signs (c, s). (The response s will be different for each TTP, but there will be only a single challenge c.) It sends all (σ, s) to the requestor, along with c, A_i, U_j, and the responses for the proofs of knowledge over v and/or the hidden attributes.
6. The requestor, who knows which TTP is used for each credential i, uses the appropriate response s out of all responses that it receives from the user when verifying the proofs of knowledge, and for each response s involving a TTP it enforces the presence of a valid signature σ from that TTP over (c, s).

Preliminaries

Lemma 1. *Let X_1 and X_2 be two identically distributed, independent discrete stochastic variables, taking values from a set of size N. Then $\Pr(X_1 = X_2) \geq \frac{1}{N}$.*

We prove the following equivalent lemma.

Lemma 2. *Let $f : \mathbb{Z}_{\leq N} \to [0, 1]$, with $\sum_{i=1}^{N} f(i) = 1$. Then $\sum_{i=1}^{N} f(i)^2 \geq \frac{1}{N}$.*

Proof. We show this by induction. In the case that $N = 1$, $\sum_{i=1}^{N} f(i) = 1$ implies $f(1) = 1$, hence $f(1)^2 = 1$.

Suppose the lemma holds for $N-1$. We consider two cases: if $f(N) = 1$, then the result is immediate. Otherwise let $g : \mathbb{Z}_{\leq N-1} \to [0,1]$ be defined through $g(i) = \frac{f(i)}{1-f(N)}$. Calculating, we find:

$$\begin{aligned}\sum_{i=1}^{N} f(i)^2 &= (1-f(N))^2 \sum_{i=1}^{N-1} g(i)^2 + f(N)^2 \\ &\geq (1-f(N))^2 \frac{1}{N-1} + f(N)^2\end{aligned}$$

Considering this last line as a function in the variable $f(N)$, by the combination of Fermat's theorem on stationary points and the extreme value theorem, it attains its minimum either at $f(N) = 0$, $f(N) = 1$, or when $-2(1-f(N))\frac{1}{N-1} + 2f(N) = 0$. The case $f(N) = 1$ we already dealt with above. For $f(N) = 0$, the result is immediate. For the remaining possibility, note that this implies $f(N) = 1/N$, which yields

$$\begin{aligned}\sum_{i=1}^{N} f(i)^2 &\geq \left(\frac{N-1}{N}\right)^2 \frac{1}{N-1} + \frac{1}{N^2} \\ &= \frac{1}{N}\left(\frac{N-1}{N} + \frac{1}{N}\right) = \frac{1}{N}.\end{aligned}$$ □

From this, Lemma 1 follows directly.

References

[22a] *IRMA*. https://irma.app. 2022.

[22b] *IRMA technical documentation*. https://irma.app/docs. 2022.

[ASM06] M. H. Au, W. Susilo, and Y. Mu. "Constant-Size Dynamic k-TAA". In: *Security and Cryptography for Networks*. Ed. by R. De Prisco and M. Yung. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 111–125. ISBN: 978-3-540-38081-8.

[CDL16] J. Camenisch, M. Drijvers, and A. Lehmann. "Anonymous Attestation Using the Strong Diffie Hellman Assumption Revisited". In: *Trust and Trustworthy Computing*. Ed. by M. Franz and P. Papadimitratos. https://eprint.iacr.org/2016/663. Cham: Springer International Publishing, 2016, pp. 1–20. ISBN: 978-3-319-45572-3.

[FS87] A. Fiat and A. Shamir. "How To Prove Yourself: Practical Solutions to Identification and Signature Problems". In: *Advances in Cryptology – CRYPTO' 86*. Ed. by A. M. Odlyzko. Vol. 263. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1987, pp. 186–194. DOI: 10.1007/3-540-47721-7_12.

[IBM12] IBM Research Zürich Security Team. *Specification of the Identity Mixer Cryptographic Library, version 2.3.4*. Tech. rep. IBM Research, Zürich, Feb. 2012.

[Sch90] C. Schnorr. “Efficient Identification and Signatures for Smart Cards”. In: *Advances in Cryptology — EUROCRYPT ’89*. Ed. by J.-J. Quisquater and J. Vandewalle. Vol. 434. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1990, pp. 688–689. ISBN: 978-3-540-53433-4. DOI: `10.1007/3-540-46885-4_68`.