Secret Handshakes with Revocation Support

Alessandro Sorniotti[1,2] and Refik Molva[2]

[1] SAP Research,
805, Docteur Maurice Donat,
06250 Mougins, France,
[2] Institut Eurécom
2229, Route des Crêtes
06560 Valbonne, France
first.last@eurecom.fr

Abstract. Secret Handshake is becoming an ever more popular research subject in the field of privacy-preserving authentication protocols. Revocation of credentials in Secret Handshakes is a difficult challenge, as it mixes the conflicting requirements of tracing revoked users and of the untraceability and unlinkability of legitimate protocol players. The schemes proposed in the literature are either limited versions of secret handshake supporting revocation, or they support more complete versions of secret handshake with no possibility of introducing revocation. In this paper we present a simple protocol that allows a user to prove to a verifier possession of a credential. Credentials can be revoked simply by publishing a value in a revocation list. This protocol is extremely flexible, as with it, we can achieve revocation for each of the different nuances of Secret Handshakes known in the literature. We prove the security of the new scheme without random oracles.

1 Introduction

The topic of secret handshakes is gaining momentum in research, as evidenced by the number of recent publications on the subject [1, 21, 10, 23, 12]. The concept has been introduced by Balfanz and colleagues in [5]; secret handshake protocols can be used by two parties to share a key only if they both belong to a common secret group. The protocol makes sure that an outsider, or an illegitimate group member does not learn anything by interacting with a legitimate user or eavesdropping on protocol exchanges.

The original protocol proposed by Balfanz et al. [5] suffered from a number of shortcomings, namely it only allows users to prove membership to the same group, and it requires multiple credentials, as exchanged credentials are traceable over multiple executions. These shortcomings have been fixed by subsequent schemes that support the reuse of credentials and allow to match properties different from a user's own [10, 25, 15, 14, 21].

In spite of these enhancements, Secret Handshakes still suffer from significant limitations due to the inherent difficulty of supporting revocation of credentials. Revocation indeed represents an interesting challenge: on the one hand, protocol messages need to be untraceable; on the other, revocation requires means of tagging credentials in order to single out the revoked ones and refuse any interaction with users bearing them.

So far this problem has still not been solved: among the schemes presented in the literature, there are either limited versions of secret handshake schemes that support revocation [27, 5, 10], or other schemes that support more complete versions of secret handshake with no possibility of introducing revocation, at least not without radical changes to the protocol [1, 23].

The contributions of this paper are manifold: (i) we present a novel scheme called RevocationMatching, a building block for addressing in a comprehensive way all Secret Handshake scenarios known in the literature, with the additional feature of revocation; (ii) with RevocationMatching we can build each of the different "flavors" of secret handshake, own-group membership secret handshakes [5, 10, 18, 25, 27], secret handshakes with dynamic matching [1] or secret handshake with dynamic controlled matching [23], adding revocation support to each. In addition, (iii) our scheme supports the existence of multiple CAs, which also represent an interesting advancement to the state of the art of Secret Handshakes.

In all the different schemes that we propose, credentials can be efficiently revoked. After their revocation, credentials naturally lose their untraceability; however, as we shall see, only users authorized to match a given credential will be able to trace the revoked credential; for other users, the credential will still be unlinkable and untraceable. We analyze the security of RevocationMatching and of the derived Secret Handshake schemes *without* random oracles, by reduction to intractable problems.

We organize the rest of this paper as follows: in Section 2 we present the related work and underline the contributions of this paper; in Section 3 we present the preliminaries and we give an intuitive overview of how the solution is designed. Section 4 illustrates RevocationMatching, whereas Section 5 shows how it can be used to address all known Secret Handshake protocols in the state of the art, with revocation support. Section 6 analyzes the security of the scheme and Section 7 concludes with some outlook on future work.

2 Related Work and Contribution

The goal of this Section is to walk the reader through the related work carried out in the field of secret handshakes, highlighting the different existing protocols and positioning the contribution of this paper.

Secret Handshakes have been introduced by Balfanz et al. [5] as a mechanism devised for two users to simultaneously prove to each other possession of a property, for instance membership to a certain group. The ability to prove and verify is strictly controlled by a certification authority, that issues property credentials and matching references respectively allowing to prove to another user, and to verify another user's, possession of a property. Balfanz' original scheme, as many other schemes in the literature, only supports proving and verifying membership to the same group: for this reason, we shall call this family of schemes *own-group membership secret handshakes*. The proposed scheme supports revocation, but has a number of drawbacks, for instance the fact that it relies on one-time credentials to achieve untraceability. After this seminal work, many papers have further investigated the subject of secret handshake, considerably advancing the state of the art. The work by Castelluccia et al. [10] has shown how, under some specific requirements (namely CA-obliviousness), secret handshakes can be obtained from PKI-enabled encryption schemes. Other schemes have followed this approach [25, 15] offering similar results, albeit with different nuances of unlinkability. Almost all the schemes in this family support revocation of credentials; however the functionalities offered are limited to proving and verifying membership to a common group.

[14, 21] show how leveraging on authenticated key exchanges, we can build Secret Handshakes. Shin and Gligor [21] use password-authenticated key exchanges to establish a common key, provided that there is a match of one common interest or "wish". This protocol is very similar to the family of *Secure Matchmaking* protocols [4, 28]. Protocols in this family support revocation but suffer from a drawback: the CA cannot exercise any control over who has the right to match which property. While this is a good feature in matchmaking-like scenarios, it is an undesirable feature in more sensitive scenarios (see [23]). In addition, schemes in this family are again limited to proof and verification of membership to a common group.

An advancement on this front has recently been put forward by Ateniese et al. [1], who have introduced *dynamic matching* and Sorniotti et al. [23] who have proposed the similar concept of *dynamic controlled matching*. Both schemes allow more flexible types of handshakes: members of different groups, or more generally, users holding credentials for different properties, can conduct a successful secret handshake if credentials match the other user's matching references. The difference between the two schemes is the control that the CA retains over the matching ability. Both schemes are extremely flexible, covering the functionalities of own-group membership secret handshakes and adding dynamic matching; however, neither of them support revocation of credentials.

A related topic is represented by oblivious signature-based envelopes (OSBEs), introduced by Li et al. in [16]; using OSBE, a sender can send an envelope to a receiver, with the assurance that the receiver will only be able to open it if he holds the signature on an agreed-upon message. Nasserian and Tsudik in [19] argue – with no proofs – that two symmetric instances of OSBE may yield a Secret Handshake: however OSBE does not consider unlinkability and untraceability, as it requires the explicit agreement on a signature beforehand. Camenisch et al. have shown in [9] how dynamic accumulators [20, 6] can be used to achieve efficient revocation for anonymous credentials. However dynamic accumulators, quoting Balfanz [5], are ill-suited for secret handshakes, mainly due to the fact that when a verifier has checked that a prover's witness belongs to the accumulator, he has already disclosed the prover's affiliation and can then selfishly refuse to reveal his own witness, or can reveal a fake one. Turning accumulator based asymmetric membership verification into symmetric handshakes is indeed an interesting open challenge. In addition, it is not possible to control who can verify what, dynamic matching cannot be supported, and finally, tracing traitors is not feasible.

The scheme that we present in this paper can reproduce the functionalities of the different families of secret handshakes that we have discussed in this Section, with the built-in revocation support. In addition, our scheme supports as well the existence of multiple CAs, which represent an interesting advancement to the state of the art of Secret Handshakes.

3 An overview of the solution

In this Section we give the reader an insight on the reasons and choices behind the actual design of the scheme.

At first, let us describe the notation used in the sequel of the paper. Given a security parameter k, let $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ be groups of order q for some large prime q, where the bit-size of q is determined by the security parameter k. Our scheme uses a computable, non-degenerate bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ for which the *Symmetric External Diffie-Hellman (SXDH)* problem is assumed to be hard. The SXDH assumption in short allows for the existence of a bilinear pairing, but assumes that the Decisional Diffie-Hellman problem is hard in both $\mathbb{G}_1$ and $\mathbb{G}_2$ (see [1] for more details).

We then describe how we represent strings into group elements. Following [7, 26], let $\tilde{g} \overset{R}{\leftarrow} \mathbb{G}_2$; let us also choose $n+1$ random values $\{y_i\}_{i=0}^{n} \overset{R}{\leftarrow} \mathbb{Z}_q^*$; we assign $\tilde{g}_0 = \tilde{g}^{y_0}, \tilde{g}_1 = \tilde{g}^{y_1}, \ldots, \tilde{g}_n = \tilde{g}^{y_n}$. If $v \in \{0,1\}^n$ is an n-bit string, let us define $h(v) = y_0 + \sum_{i \in V(v)} y_i \in \mathbb{Z}_q^*$, where $V(v)$ represents the set of indexes i for which the ith bit of v is equal to 1. We also define $\tilde{H}(v) = \tilde{g}_0 \prod_{i \in V(v)} \tilde{g}_i = \tilde{g}^{h(v)} \in \mathbb{G}_2$.

Our starting objective is to design a scheme that helps a prover convince a verifier that she owns the credential for a property; however, the verification will be successful only for entitled verifiers. The exchange must satisfy the standard security requirements for Secret Handshakes, detector and impersonator resistance and untraceability of properties and identities. On top of this, we also want to support revocation of credentials. To this end, we need some means of secretly "labeling" each credential, so that we can later on reveal the label and use it as a handle to refuse handshake instances embedding it. In this section we try to walk the reader through the design of the solution.

Let us assume that g and $\tilde{g}$ are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. Also, $t \in \mathbb{Z}_q^*$ is a master secret. Then, given a property p, matching references can be formed as $\tilde{g}^{h(p)t} \in \mathbb{G}_2$ and given to verifiers; in order to successfully authenticate as a possessor of property p, a prover must then prove knowledge of $g^{h(p)t} \in \mathbb{G}_1$. However, instead of simply giving that value to the prover, we pick a random value $x \in \mathbb{Z}_q^*$, different for every credential, and give x and $g^{(x+h(p)t)}$ to the prover. $g^{(x+h(p)t)}$ is the credential and x is the aforementioned tag, called *identification handle* in the rest of this paper, used to identify credentials that need to be revoked.

Then, a prover can be authenticated by a verifier as follows: the verifier sends a challenge $\hat{e}\left(g, \tilde{g}\right)^m$ and receives $\left\langle g^{r(x+h(p)t)}, g^r \right\rangle$ from the prover, where r is a random number, used by the prover to salt the handshake message. The prover can compute $K = \left(\hat{e}\left(g, \tilde{g}\right)^m\right)^{rx}$ and the verifier can compute $K' = \left(\hat{e}\left(g^{r(x+h(p)t)}, \tilde{g}\right) / \hat{e}\left(g^r, \tilde{g}^{h(p)t}\right)\right)^m$; if the authentication is successful, K and K' are the same.

If the credential is to be revoked at some point, all we need to do is reveal $\tilde{g}^x$, called *revocation handle* in the rest of the paper. This way, the verifier can verify if the credential used by the prover has been revoked, by checking if $\hat{e}\left(g^{r(x+h(p)t)}, \tilde{g}\right) = \hat{e}\left(g^r, \tilde{g}^{h(p)t} \cdot \tilde{g}^x\right)$ holds.

Two challenges arise: first, it should be impossible to use the value x in order to trace credentials before they have been revoked; and second, a user should be forced to send credentials unmodified. The solution presented above respects the privacy of users: prior to the revocation of a given credential, an attacker cannot use the identification handle to link two different instances of the handshake to the same user: it is easy to show that linking the same x through subsequent instances of the protocol, is equivalent to solving DDH in $\mathbb{G}_1$.

However this solution still does not force the attacker to send his credentials unmodified, which would imply that an attacker can circumvent revocation. In order to prevent this attack, we also introduce another public parameter $W = g^w$, where $w \overset{R}{\leftarrow} \mathbb{Z}_q^*$ is kept secret. Each credential is multiplied by a different random number, for instance $g^{z(x+h(p)t)}$; in addition, the prover also receives $\tilde{g}^{z^{-1}}$ and $\tilde{g}^{(zw)^{-1}}$. The verifier then computes $K = \left(\hat{e}\left(g^{rz(x+h(p)t)}, \tilde{g}^{z^{-1}}\right)/\hat{e}\left(g^r, \tilde{g}^{h(p)t}\right)\right)^m$. In addition we require verifier to also verify that $\hat{e}\left(g, \tilde{g}^{z^{-1}}\right) = \hat{e}\left(W, \tilde{g}^{(zw)^{-1}}\right)$.

The protocol introduced in the next Section is not very different from the simple one that we proposed here. Among the modifications, we include an additional random number used to also salt the terms $\tilde{g}^{z^{-1}}$ and $\tilde{g}^{(zw)^{-1}}$, which would otherwise not be randomized and open up to tracing attacks.

4 RevocationMatching: a Prover-Verifier Scheme with revocation capabilities

RevocationMatching is a protocol wherein a prover can convince a verifier that she owns a property. The active parties are essentially users, that can behave as provers and verifiers, and a trusted entity that we will call certification authority (CA). Provers receive from the CA credentials for a given property, allowing them to convince a verifier that they possess that property. Verifiers in turn receive from the CA credentials matching references for a given property, which allow them to verify the possession of that property. In case of compromised credentials, the CA adds a value called revocation handle to a publicly available revocation list: this way, verifiers may refuse to interact with users bearing revoked credentials.
RevocationMatching consists of the following algorithms and protocols:

- Setup: according to the security parameter k, the CA chooses $g, \tilde{g}$, where g is a random generator of $\mathbb{G}_1$ and $\tilde{g}$ of $\mathbb{G}_2$. The CA sets $e = \hat{e}(g, \tilde{g})$. The CA also picks $w, t \overset{R}{\leftarrow} \mathbb{Z}_q^*$ and sets $W \leftarrow g^w$ and $T \leftarrow \tilde{g}^t$. Finally the CA picks $\{y_i\}_{i=0}^n \overset{R}{\leftarrow} \mathbb{Z}_q^*$ and assigns $g_0 \leftarrow g^{y_0}, g_1 \leftarrow g^{y_1}, \ldots, g_n \leftarrow g^{y_n}$ and $\tilde{g}_0 \leftarrow \tilde{g}^{y_0}, \tilde{g}_1 \leftarrow \tilde{g}^{y_1}, \ldots, \tilde{g}_n \leftarrow \tilde{g}^{y_n}$; this way, given a string v, $H(v) = g^{h(v)}$ and $\tilde{H}(v) = \tilde{g}^{h(v)}$. The system's public parameters are $\{q, \mathbb{G}_1, \mathbb{G}_2, g, \tilde{g}, W, T, g_0, \ldots, g_n, \tilde{g}_0, \ldots, \tilde{g}_n, \hat{e}, e\}$. The values $w, t, y_0, \ldots, y_n$ are instead kept secret by the CA;
- Certify: upon user request, the CA verifies that the supplicant user $u \in \mathcal{U}$ possesses the property $p \in \mathcal{P}$ she will later claim to have during the protocol execution; after a successful check, the CA issues to u the appropriate credential, which is made of two separate components: an identification handle, later used for revocation, and the actual credential. To hand out the identification handle for a given pair (u, p), the CA picks the identification handle $x_{u,p} \overset{R}{\leftarrow} \mathbb{Z}_q^*$, randomly drawn upon each query, and gives it to the supplicant user. The CA then forms the credential as a tuple $cred_{u,p} = \langle C_{u,p,1}, C_{u,p,2}, C_{u,p,3} \rangle$ where $C_{u,p,1} = g^{z(x_{u,p}+h(p)(t+h(p)))}$, $C_{u,p,2} = \tilde{g}^{z^{-1}}$ and $C_{u,p,3} = \tilde{g}^{(zw)^{-1}}$, where $z \in \mathbb{Z}_q^*$ is randomly drawn upon each query. The user can verify the validity of the credential by checking that $\hat{e}(C_{u,p,1}, C_{u,p,2}) = \hat{e}(g^{x_{u,p}}, \tilde{g}) \cdot \hat{e}(H(p), T \cdot \tilde{H}(p))$;
- Grant: upon a user's request, the CA verifies that – according to the policies of the system – user u is entitled to verify that another user possesses property $p \in \mathcal{P}$. If the checking is successful, the CA issues the appropriate matching reference $match_p = \left(T \cdot \tilde{H}(p)\right)^{h(p)}$; the user verifies that $\hat{e}(g, match_p) = \hat{e}(H(p), T \cdot \tilde{H}(p))$;
- Authenticate: let A be a prover and B a verifier. A has $cred_{A,p_1}$ and x_{A,p_1} to prove possession of property p_1; B holds $match_{p_2}$ to detect property p_2. The protocol proceeds as follows:
  1. B picks $m \overset{R}{\leftarrow} \mathbb{Z}_q^*$ and sends e^m to A

2. A picks $r, s \xleftarrow{R} \mathbb{Z}_q^*$ and sends B the tuple $\left\langle g^r, \left(C_{A,p_1,1}\right)^{rs}, \left(C_{A,p_1,2}\right)^{s^{-1}}, \left(C_{A,p_1,3}\right)^{s^{-1}} \right\rangle$. A locally computes $K = \left(e^m\right)^{r x_{A,p_1}}$
3. B checks whether

$$\hat{e}\left(W, \left(C_{A,p_1,3}\right)^{s^{-1}}\right) = \hat{e}\left(g, \left(C_{A,p_1,2}\right)^{s^{-1}}\right) \tag{1}$$

and locally computes

$$K = \left(\frac{\hat{e}\left(\left(C_{A,p_1,1}\right)^{rs}, \left(C_{A,p_1,2}\right)^{s^{-1}}\right)}{\hat{e}\left(g^r, match_{p_2}\right)} \right)^m \tag{2}$$

At the end of the protocol, A and B share the same key K if $p_1 = p_2$.

– Revoke: if the credential for property p of user $u \in \mathcal{U}$ is to be revoked, the CA adds the so-called *revocation handle* $rev_{u,p} = \tilde{g}^{x_{u,p}}$ to a publicly available revocation list L_{rev}. Notice the tight relationship between the identification handle $x_{u,p}$ and the corresponding revocation handle $rev_{u,p} = \tilde{g}^{x_{u,p}}$.
Let us assume that a given user A is using the protocol to convince user B she owns a property; B receives $\left\langle g^r, \left(C_{A,p,1}\right)^{rs}, \left(C_{A,p,2}\right)^{s^{-1}}, \left(C_{A,p,3}\right)^{s^{-1}} \right\rangle$ from A. B behaves as follows: first, she performs the check of Equation 1; then, before computing the key K, she verifies whether A is using a revoked credential by checking if the following identity

$$\hat{e}\left(\left(C_{A,p,1}\right)^{rs}, \left(C_{A,p,2}\right)^{s^{-1}}\right) = \hat{e}\left(g^r, match_p \cdot rev\right) \tag{3}$$

is verified with any of the revocation handles rev in the list L_{rev}. If the check is successful, B discards the current handshake instance. Notice that if B does not have the correct matching reference for the received credential, Revoke would fail altogether; however, so would Authenticate, in which case, the receiving user would discard the handshake instance anyway. It is clear that after revocation, credentials can be traced *only* by users that possess the matching reference for the property object of that credential; these users were already potentially able to match the given credential. For other users, past and future transcripts of handshake instances produced from that credentials are still untraceable and unlinkable.

5 Building Secret Handshakes

In this section, we show how RevocationMatching can be used to build a two-party Handshake scheme. This scheme helps two users share a key in case of simultaneous successful matching of properties, and gives no clue about one another's properties otherwise. The resulting scheme, contrary to many secret handshake schemes in the state of the art, does not only allow users to verify if they possess the same property (e.g. if they belong to the same secret group). Our scheme also supports dynamic matching, as introduced by Ateniese et al. in [1]: with dynamic matching, users can match properties different from the ones they possess. For instance, a member of CIA and a member of MI5 can successfully authenticate with a secret handshake. Additional use-cases are described in [23].

We actually consider two different flavors of dynamic matching, thus proposing a total of three different schemes (four considering the multiple CA scenario): in the first scheme, credentials and matching references are issued by the certification authority; this way, the CA retains the control over who can prove what and who can verify what. In the second scheme instead, while credentials are still issued by the certification authority, matching references can be computed by users without any required intervention by the CA. We will refer to the first scheme as Secret Handshake with Dynamic Controlled Matching, and to the second one as Secret Handshake with Dynamic Matching.

Alice :	pick $r, s, m \xleftarrow{R} \mathbb{Z}_q^*$
Alice $\longrightarrow$ Bob :	$\left\langle g^r, (C_{A,p_1,1})^{rs}, (C_{A,p_1,2})^{s^{-1}}, (C_{A,p_1,3})^{s^{-1}}, e^m \right\rangle$
Bob :	pick $r', s', m' \xleftarrow{R} \mathbb{Z}_q^*$
Bob $\longrightarrow$ Alice :	$\left\langle g^{r'}, (C_{B,p_2,1})^{r's'}, (C_{B,p_2,2})^{s'^{-1}}, (C_{B,p_2,3})^{s'^{-1}}, e^{m'} \right\rangle$
Alice :	check that Equation 1 holds, otherwise abort
Alice :	check that Equation 3 is not satisfied with any $rev \in L_{rev}$, otherwise abort
Alice :	compute $K_1 = \left(e^{m'}\right)^{rx_{A,p_1}}$
Alice :	compute $K_2 = \left(\frac{\hat{e}\left((C_{B,p_2,1})^{r's'}, (C_{B,p_2,2})^{s'^{-1}} \right)}{\hat{e}\left(g^{r'}, match_{p_2}\right)} \right)^m$
Bob :	check that Equation 1 holds, otherwise abort
Bob :	check that Equation 3 is not satisfied with any $rev \in L_{rev}$, otherwise abort
Bob :	compute $K_1 = \left(\frac{\hat{e}\left((C_{A,p_1,1})^{rs}, (C_{A,p_1,2})^{s^{-1}} \right)}{\hat{e}\left(g^{r}, match_{p_1}\right)} \right)^{m'}$
Bob :	compute $K_2 = (e^m)^{r'x_{A,p_1}}$
Alice $\longleftrightarrow$ Bob:	mutual proof of knowledge of K_1 and K_2

Fig. 1. Secret Handshake with Dynamic Controlled Matching

5.1 Secret Handshake with Dynamic Controlled Matching

In this scheme, users receive credentials and matching references from the certification authority. Matching references can only be computed by the CA. Notice that they do not necessarily refer to the same property as credentials; this way we effectively achieve Secret Handshake with dynamic controlled matching [23]. However, notice that the CA may enforce the policy by which only users owning a given property will receive the matching reference for it, thus achieving own-group membership secret handshakes too [5, 10, 18, 25, 27].

The secret handshake is achieved by running two symmetric instances of RevocationMatching, wherein each of the two users plays in turn the role of prover and verifier. Each user will then end up with two keys, one computed in the role of prover and the other one computed in the role of verifier. We borrow the idea of computing two separate keys at each user's side from Ateniese et al. [1]. To seal the handshake, the two users have to prove one another knowledge of both keys simultaneously, for instance trying to establish a secure channel with a key resulting from the hash of the concatenation of both keys.

Let us assume that two users, Alice and Bob, want to perform a Secret Handshake and share a key if the Handshake is successful. Alice owns the tuple $\langle cred_{A,p_1}, match_{p_2},\ x_{A,p_1} \rangle$ and Bob owns $\langle cred_{B,p_2}, match_{p_1}, x_{B,p_2} \rangle$. Figure 1 shows how the handshake is carried out.

At the completion of the protocol, Alice and Bob share the same keypair if and only if each user's credential matches the other user's matching reference. If not, one of the two keys, or both, will be different. By requiring them to prove to one another knowledge of both keys simultaneously, either both users learn of a mutual matching, or they do not learn anything at all. In particular, they do not learn – in case of a failed handshake – if just one of the two matchings have failed, and if so which one, or if both did fail.

5.2 Multiple CA support

The scheme also supports the existence of multiple CAs. Multiple CAs may be a requirement when the properties at stake are – for example – membership to different secret agencies that do not want to delegate the execution of Certify, Grant and Revoke for security reasons. In a multiple CA scenario, a handshake can be successful even in hybrid situations in which Alice has a credential for property p_1 issued from CA_1 and a matching reference for property p_2 issued from CA_2 and Bob has a credential for property p_2 issued from CA_2 and a matching reference for property p_1 issued from CA_1.

A multiple CA scenario can be supported as follows: one of the CAs picks $\{q, \mathbb{G}_1, \mathbb{G}_2, g, \tilde{g}, T, g_0, \ldots, g_n, \tilde{g}_0, \ldots, \tilde{g}_n, \hat{e}, e\}$; the values $\{y_i\}_{i=0}^{n} \overset{R}{\leftarrow} \mathbb{Z}_q^*$ and t are shared among all the CAs. Then the CAs jointly generate $W = g^w$ and $\tilde{g}^{w^{-1}}$, such that w is unknown: the CAs can achieve this either by using a trusted dealer or by performing the following joint computation: they organize themselves in a chain; the first node A picks $a \overset{R}{\leftarrow} \mathbb{Z}_q^*$ and sends to B, the next node, g^a and $\tilde{g}^{a^{-1}}$; B in turn picks $b \overset{R}{\leftarrow} \mathbb{Z}_q^*$ and sends to the next node g^{ab} and $\tilde{g}^{(ab)^{-1}}$ and so forth until the last node is reached.

Finally, each CA picks a secret value $t_{CA} \overset{R}{\leftarrow} \mathbb{Z}_q^*$ and publishes $T_{CA} \leftarrow g^{t_{CA}}$; the public parameters are $\{q, \mathbb{G}_1, \mathbb{G}_2, g, \tilde{g}, W, T, T_{CA_1}, \ldots, T_{CA_n}, g_0, \ldots, g_n, \tilde{g}_0, \ldots, \tilde{g}_n, \hat{e}, e\}$; each CA keeps the values $\{y_i\}_{i=0}^{n} \overset{R}{\leftarrow} \mathbb{Z}_q^*$, t, t_{CA} and $\tilde{g}^{w^{-1}}$ secret.

A given CA forms credentials as $cred_{u,p} = \langle C_{u,p,1}, C_{u,p,2}, C_{u,p,3} \rangle$ where $C_{u,p,1} = W^{z(x_{u,p}+h(p)t_{CA}(t+h(p)))}$, $C_{u,p,2} = \left(\tilde{g}^{w^{-1}}\right)^{z^{-1}}$ and $C_{u,p,3} = \tilde{g}^{z^{-1}}$; the matching reference is formed as $match_p = \left(T \cdot \tilde{H}(p)\right)^{h(p)t_{CA}}$. The check of Equation 1 becomes $\hat{e}\left(W, (C_{u,p,2})^{s^{-1}}\right) = \hat{e}\left(g, (C_{u,p,3})^{s^{-1}}\right)$. Users cannot any longer verify the correctness of credentials and matching reference; for this reason, the issuing CA also gives the value $H(p)^{t_{CA}}$ to the supplicant user; the user can verify that $\hat{e}\left(H(p)^{t_{CA}}, \tilde{g}\right) = \hat{e}\left(T_{CA}, \tilde{H}(p)\right)$ and then use $H(p)^{t_{CA}}$ instead of $H(p)$ in the verification equations.

Revocation handles must be published in a common revocation list, where all CAs publish revocation handles. The list is common, public, and there is nothing that gives away which CA is behind which revocation value. For the rest, the scheme behaves as before.

5.3 Secret Handshake with Dynamic Matching

In the scheme described in this Section, each user can freely compute matching references of his choice: this way we effectively achieve dynamic matching, in the sense first defined by Ateniese et al. [1]. The secret handshake with dynamic matching is essentially equal to the scheme introduced in Section 5.1, with a substantial difference: the parameter T is now equal to $\tilde{g}$, and consequently, $t = 1$; as a consequence, we simplify $C_{u,p,1} = g^{z(x_{u,p}+h(p))}$ and $match_p = \tilde{H}(p) = \tilde{g}^{h(p)}$. Notice that now users are able to create matching references at their will.

With this scheme, two users with valid credentials can interact expressing *wishes* on the property certified by the other user's credential; wishes are represented by self-generated matching references. Both users at the end of the protocol share a common key pair if they both own credentials for the property expected by the other user. Notice that this change makes it impossible to support multiple CAs, as specified in Section 5.2.

Revocation handles are also formed differently, as $rev_{u,p} = \tilde{g}^{x_{u,p}+h(p)}$. Consequently when a user B receives from user A $\left\langle g^r, (C_{A,p,1})^{rs}, (C_{A,p,2})^{s^{-1}}, (C_{A,p,3})^{s^{-1}} \right\rangle$, B verifies whether A is using a revoked credential by checking if $\hat{e}\left((C_{A,p,1})^{rs}, (C_{A,p,2})^{s^{-1}}\right) = \hat{e}\left(g^r, rev\right)$ is verified with any of the revocation handles rev in the list L_{rev}. If the check is successful, B discards the current handshake instance and declines any further interaction.

The change in how the revocation handle is constructed can be explained through the fact that in this case every user has the right to match any property. A revoked credential therefore loses its untraceability to every other user. However, the revocation handle still does not reveal anything about the nature of the certified property.

6 Security Analysis

Before proceeding further, we state two well-known hard problems:

Definition 1 (*Decisional Diffie-Hellman* Problem). *We say that the Decisional Diffie-Hellman Problem (DDH) is hard if, for all probabilistic, polynomial-time algorithms $\mathcal{B}$,*

$$\mathsf{AdvDDH}_B := Pr[\mathcal{B}(g, g^a, g^b, g^x) = \textit{\texttt{true}} \text{ if } x = ab] - \tfrac{1}{2}$$

is negligible in the security parameter. We assume a random choice of $g \in \mathbb{G}_1$, a, $b \in \mathbb{Z}_q^*$; x is equal to ab with probability $\frac{1}{2}$ and is otherwise equal to a random value in $\mathbb{Z}_q^*/\{ab\}$ with the same probability.

Definition 2 (*Bilinear Decisional Diffie-Hellman* Problem). *We say that the Bilinear Decisional Diffie-Hellman Problem (BDDH) is hard if, for all probabilistic, polynomial-time algorithms $\mathcal{B}$,*

$$\mathsf{AdvBDDH}_B := Pr[\mathcal{B}(g, g^a, g^b, g^c, \tilde{g}, \tilde{g}^a, \tilde{g}^b, \tilde{g}^x) = \textit{\texttt{true}} \text{ if } x = abc] - \tfrac{1}{2}$$

is negligible in the security parameter. We assume a random choice of $g \in \mathbb{G}_1$, $\tilde{g} \in \mathbb{G}_2$ and a, b, $c \in \mathbb{Z}_q^*$; x is equal to abc with probability $\frac{1}{2}$ and is otherwise equal to a random value in $\mathbb{Z}_q^*/\{abc\}$ with the same probability.

We also introduce the following new intractability assumption; we will give evidence of its hardness in Appendix A. In demonstrating the complexity assumption, we follow the approach presented by Victor Shoup in [22] and extensively used by the research community [2, 8, 17]. As an example, the well known SDH assumption was thus proved by Boneh and Boyen in [8].

Definition 3. *[SM Problem] Let $w, y, m \in \mathbb{Z}_q^*$, let g be a generator of $\mathbb{G}_1$ and $\tilde{g}$ be a generator of $\mathbb{G}_2$. Let oracle $O_{w,y}(\cdot)$ take input $x \in \mathbb{Z}_q^*$ and produce output $g^{z(x+y)}$, $\tilde{g}^{z^{-1}}$ and $\tilde{g}^{(zw)^{-1}}$ where z is randomly drawn from $\mathbb{Z}_q^*$ upon each oracle query. We say that the SM Problem is hard if, for all probabilistic, polynomial-time algorithms $\mathcal{A}$,*

$$\mathsf{AdvSM}_{\mathcal{A}} := Pr[\mathcal{A}(g, g^w, \tilde{g}, \tilde{g}^y, \tilde{g}^m, O_{w,y}) = a, a^{s(x_*+y)}, \tilde{g}^{(sw)^{-1}}, \tilde{g}^{(s)^{-1}}, \hat{e}\,(a, \tilde{g})^{mx_*}]$$

such that $(x_*) \notin \mathcal{O}$, is negligible in the security parameter; $a \in \mathbb{G}_1$. $\mathcal{O}$ is the set of queries $\mathcal{A}$ makes to oracle $O_{w,y}$. This probability is taken over random choice of $g \in \mathbb{G}_1$, $\tilde{g} \in \mathbb{G}_2$, and $w, y, m \in \mathbb{Z}_q^*$. $a \in \mathbb{G}_1$ can be chosen freely by the adversary.

Intuitively, the assumption tells that it is unfeasible to compute a tuple $\left\langle g^{s(x_*+y)}, \tilde{g}^{s^{-1}}, \tilde{g}^{(sw)^{-1}} \right\rangle$ for a new value x_* and prove knowledge of it, yet having an oracle that can do so for any query. The new assumption is generic enough to be of independent interest, for instance to realize signature protocols or oblivious signature-based envelopes. Our assumption could have had a simpler formulation had we chosen not to embed the proof of knowledge of x_* in it.

6.1 Security Analysis of RevocationMatching

In this section we analyze the security requirements of RevocationMatching. We base our security analysis on the security definitions and attacker model presented by Balfanz and colleagues in [5] and Ateniese and colleagues in [1].

At first, we briefly recapitulate some preliminary definitions: a complete description of the security definition and attacker model can be found in [5, 1]. A *protocol instance* is the interaction of two users according to the rules of the protocol. We say that during a protocol instance a prover sends a *handshake tuple*. The handshake tuple *contains* a property, in that it is formed out of a credential certifying possession of a property, which is then the *object* of the protocol instance. At the end of the Authenticate algorithm, users are asked to prove to one another knowledge of a locally computed key. It is a necessary prerequisite that the proof of knowledge does not – in any way – leak the actual value of the key. If the computed key is the same for both user, we say that the prover has *proved* to or *convinced* a verifier that she owns the property object of the handshake; and that the verifier has *verified* or *detected* the presence of a property within a handshake tuple during a protocol instance.

To analyze the security of RevocationMatching, we identify four different objectives that an attacker might have. An attacker may:

– **detection**: try as a verifier to detect a prover's property without the appropriate matching reference;
– **impersonation**: try as a prover to convince a verifier that she possesses a given property without the appropriate property credential;
– **linking**: try to link different protocol executions to a given user;
– **tracing**: try to link different protocol executions to a given property;

The sequel of this Section analyzes RevocationMatching with respect to these four requirements. Appendix B presents proofs of the claims made in this Section. Notice that the proofs do not rely on random oracles.

Untraceability Consider an adversary $\mathcal{A}$ whose goal is to check if two handshake tuples contain the same property, without owning the legitimate matching reference; an adversary with this ability can link together the different users that own credentials for a given property. In order to be general enough we consider an active adversary that engages in protocol executions; this adversary clearly also includes a passive one who just eavesdrops protocol instances.
To capture the attacker we define a game called TraceProperty. TraceProperty develops as follows:

– **Setup**: during the setup phase, the challenger generates the parameters of the system;
– **Query**: during the query phase, $\mathcal{A}$ can receive valid credentials, matching references and revocation handles, and can engage in RevocationMatching protocol execution with legitimate users;
– **Challenge**: then the challenger randomly chooses two properties p_1 and p_2 and sends $\mathcal{A}$ two handshake tuples, one for property p_1 and the other for property p_2; both properties have not been object of a query in the previous phase; $\mathcal{A}$ is then challenged to return true if $p_1 = p_2$;

Lemma 1. *Suppose that there is a probabilistic, polynomial time adversary $\mathcal{A}$ with an advantage*

$$\mathsf{AdvTraceProperty}_A := Pr[\mathcal{A} \text{ wins the game } \mathsf{TraceProperty}] - \tfrac{1}{2}$$

in the TraceProperty *game. Then a probabilistic, polynomial time algorithm $\mathcal{B}$ solves the Decisional Diffie-Hellman problem (DDH) with the same advantage.*

A proof of Lemma 1 can be found in Appendix B.1.

Unlinkability Consider an adversary $\mathcal{A}$ whose goal is to check if two handshake tuples come from the same user; an adversary with this ability can link together the same user over multiple protocol execution. In order to be general enough we consider an active adversary that engages in protocol executions; this adversary clearly also includes a passive one who just eavesdrops protocol instance.

Let us first of all notice that there are two values that can be linked to a user, the identification handle $x_{u,p}$, and z, the random number drawn at each call to Certify and used to salt the credentials. Between the two, $x_{u,p}$ is the only one that can be traced over two different handshake tuples. Indeed, tracing the value z is impossible, since over successive handshake tuples, it always appears multiplied by a different random value s chosen at random by the user himself.
To capture the attacker we define a game called TraceUser. TraceUser develops as follows:

– **Setup**: during the setup phase, the challenger generates the parameters of the system;
– **Query**: during the query phase, $\mathcal{A}$ can receive valid credentials, matching references and revocation handles, and can engage in RevocationMatching protocol execution with legitimate users;
– **Challenge**: eventually $\mathcal{A}$ receives from the challenger two handshake tuples containing the same property from users u_1 and u_2, u_1 and u_2 being chosen randomly by the challenger; $\mathcal{A}$ is challenged to return true if $u_1 = u_2$;

Lemma 2. *Suppose that there is a probabilistic, polynomial time adversary $\mathcal{A}$ with an advantage*

$$\mathsf{AdvTraceUser}_A := Pr[\mathcal{A} \textit{ wins the game } \mathsf{TraceUser}] - \tfrac{1}{2}$$

in the TraceUser *game. Then a probabilistic, polynomial time algorithm $\mathcal{B}$ solves the Decisional Diffie-Hellman problem (DDH) with the same advantage.*

A proof of Lemma 2 can be found in Appendix B.2.

Detector Resistance Let $\mathcal{A}$ be an adversary whose goal is to engage in RevocationMatching protocol instances and – acting as a verifier – to detect the prover's property, without owning the appropriate matching reference. We call detector resistance the resilience to such kind of an attacker.
To capture this kind of attack, we define a game called Detect; Detect develops as follows:

- **Setup**: during the setup phase, the challenger generates the parameters of the system;
- **Query**: the adversary $\mathcal{A}$ queries the system for an arbitrary number of tuples $\langle cred_{u_i,p_i}, match_{p_i}, x_{u_i,p_i}, rev_{u_i,p_i}\rangle$ for any given pairs $(u_i, p_i) \in \mathcal{U} \times \mathcal{P}$. She is then free to engage in RevocationMatching protocol execution with legitimate users;
- **Challenge**: $\mathcal{A}$ chooses a property p_* for which she does not own the matching reference. Also the challenger chooses a property $p_\circ$, among the ones for which the adversary does not have a matching reference. Then, challenger and adversary engage in a protocol execution; the challenger – acting as a prover – presents a credential for property $p_\circ$ and the adversary – acting as a verifier – wins the game if she can correctly whether or not $p_\circ = p_*$;

This game is very similar to TraceProperty, and the reduction to prove it is a straightforward adaptation of the one used to prove Lemma 1. Indeed an adversary $\mathcal{A}$ who has an advantage on the detection of a property without the appropriate matching reference (Detect game) can clearly link together properties over multiple handshake instances (TraceProperty game) by repetedly detecting the property in each of them and linking after the detection.

Impersonation Resistance Let $\mathcal{A}$'s goal be the impersonation of a user owning a non-revoked credential for a given property. To capture this attacker's goal we define the a game called Impersonate, which develops as follows:

- **Setup**: during the setup phase, the challenger generates the parameters of the system;
- **Query**: the adversary $\mathcal{A}$ queries the system for an arbitrary number of tuples $\langle cred_{u_i,p_i}, match_{p_i}, x_{u_i,p_i}, rev_{u_i,p_i}\rangle$ for any given pairs $(u_i, p_i) \in \mathcal{U} \times \mathcal{P}$. She is then free to engage in RevocationMatching protocol execution with legitimate users; $\mathcal{A}$ eventually decides that this phase of the game is over. The challenger then issues revocation handles for each credential handed out to the attacker in the previous phase, thus revoking them;
- **Challenge**: $\mathcal{A}$ then declares $p_* \in \mathcal{P}$ which will be the object of the challenge; $\mathcal{A}$ is then required to engage in a RevocationMatching instance with the challenger, and wins the game if she can output the correct key computed acting as a prover (notice that the same is required in [1, 5]); in order to successfully win the game, it must not be possible for the challenger to abort the handshake due to the fact that the credentials used by the attacker have been revoked;

Notice that the game covers a wide range of attacks. Recall that the attacker receives a number of credentials during the query phase. The attacker can win the game in two ways: (i) forge a brand new credential or (ii) use an old credential yet circumventing revocation. Let us set $X_{u,p} = x_{u,p} + h(p)(t + h(p))$. When the attacker is challenged, she produces the tuple $\left\langle g^r, g^{rsX_{u_*,p_*}}, \tilde{g}^{s^{-1}}, \tilde{g}^{(sw)^{-1}} \right\rangle$. If we define the set

$$Q_{\mathcal{A}} = \{X_{u,p} \in \mathbb{Z}_q^* : \mathcal{A} \textit{ has received } g^{zX_{u,p}}, \tilde{g}^{z^{-1}}, \tilde{g}^{(zw)^{-1}} \textit{ during the query phase}\}$$

then (i) implies $X_{u_*,p_*} \notin Q_{\mathcal{A}}$ and (ii) implies $X_{u_*,p_*} \in Q_{\mathcal{A}}$. We then define two different games: Impersonate1 is the aforementioned Impersonate game when $X_{u_*,p_*} \notin Q_{\mathcal{A}}$, and Impersonate2 when $X_{u_*,p_*} \in Q_{\mathcal{A}}$.

Lemma 3. *Suppose that there is a probabilistic, polynomial time adversary $\mathcal{A}$ with an advantage*

$$\mathsf{AdvImpersonate1}_A := Pr[\mathcal{A} \text{ wins the game } \mathsf{Impersonate1}]$$

in the Impersonate1 *game. Then a probabilistic, polynomial time algorithm $\mathcal{B}$ solves the SM Problem with the same advantage.*

Lemma 4. *Suppose that there is a probabilistic, polynomial time adversary $\mathcal{A}$ with an advantage*

$$\mathsf{AdvImpersonate2}_A := Pr[\mathcal{A} \text{ wins the game } \mathsf{Impersonate2}]$$

in the Impersonate2 *game. Then a probabilistic, polynomial time algorithm $\mathcal{B}$ solves the Bilinear Decisional Diffie-Hellman Problem (BDDH) with the same advantage.*

Appendixes B.3 and B.4 present the proofs of Lemmas 3 and 4, respectively. The presence of two different games to prove the same security requirement is justified by the fact that the two games cover all possible scenarios, with no possibility of hidden attacks. Indeed, either the credential produced by the attacker belongs to a set or it does not, and both cases are covered.

6.2 Security Analysis of Secret Handshake with dynamic controlled matching

In Section 5.1 we showed how to construct a protocol for Secret Handshake with dynamic controlled matching. The security requirements that have been identified for RevocationMatching in the previous Section must still hold unmodified for Secret Handshake. In the analysis of the security of secret handshake we require, as in Section 6.1, that the proof of knowledge of the keys does not – in any way – leak their actual value. In addition, we require that users prove to each other knowledge of both keys simultaneously. The same is required by other protocols in the state of the art, for instance in [1]. Examples of how this can be achieved can be found in [13].

Under these assumptions, it is straightforward how the security games and proofs devised for the latter can be adapted for Secret Handshakes: indeed untraceability and unlinkability games stay the same. As for detector and impersonation resistance, the proofs of Section 6.1 tell us that an adversary is not able to run a successful single instance of RevocationMatching acting as a rogue prover or verifier; as a consequence, given that a successful Secret Handshake requires two successful symmetric instances of RevocationMatching, an attacker acting as a rogue prover, verifier or both cannot have success in either of these two games.

6.3 Security Analysis of Secret Handshake with dynamic matching

In Section 5.3 we presented a scheme that achieves secret handshake with dynamic matching, wherein users can freely compute matching references, thus being able to match any property they want from another user. As in the previous Section, we require users to prove knowledge of both keys simultaneously, without leaking any information about them.

Let us first of all make some general considerations about the nature of the protocol. User A has the right to engage in an arbitrary number of protocol executions with any user B. If A has a legitimate credential for the wish of B (the matching reference generated by B), and guesses correctly the property object of the B's credential, then she has legitimately disclosed the challenger's property. If she is successful twice, we might say that she has been able to trace the property over two different protocol instances. Both situations are acceptable as they do not mean that a user, by simply performing secret handshake repeatedly, trying all possible matching references, will eventually discover another user's credential. Indeed, by requiring users to prove to one another knowledge of both keys, the protocol assures that if A does not have a valid credential for B's matching reference, A has no point in trying exhaustively all possible matching references to discover B's credential: one of the two instances of RevocationMatching would always fail and so would the handshake.

Among the security requirements sketched in Section 6.2, the ones related to the untraceability of identities and untraceability of properties are still the same, and the games and proofs provided in Section 6.1 still hold unchanged. As for impersonation resistance, Impersonate2 does not apply any longer, since revocation handles are now formed as $\tilde{g}^{x_{u,p}+h(p)}$; it is therefore impossible for an attacker to reuse already received credentials, yet circumventing revocation. Impersonate1 instead remains unchanged and so does its proof.

Finally, the requirement of detector resistance vanishes, since users are explicitly allowed to freely match any property by computing matching references at their will. However, as we pointed out before, a successful detection requires the user to own a credential for the other user's wish.

6.4 Security Analysis of a multiple CA scenario

Due to space restrictions we do not include the proofs of security of a multiple-CA scenario, but leave them for an extended version of the paper; nonetheless, we give here a sketch of the security analysis.

The handshake tuples produced in a multiple CA scenario are the same as in the normal case, therefore the security proofs can be easily adapted from the ones presented in the Appendixes. It remains to demonstrate that colluding CAs cannot forge credentials and matching references from a target CA_*. Forging $C_{u,p,1} = g^{zw(x_{u,p}+h(p)t_{CA_*}(t+h(p)))}$ from g^w and and $g^{t_{CA_*}}$ intuitively breaks the computational Diffie-Hellman problem; forging $match_p = (T \cdot \tilde{H}(p))^{h(p)t_{CA_*}}$ from $g^{t_{CA_*}}$ intuitively breaks the SXDH assumption, since there is no isomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$.

In addition, notice that upon a failed handshake, the information about the CA who generated the credential is not leaked; moreover, an adversary cannot trace credentials based on the CA who generated them; a similar game to TraceUser or TraceProperty can be created to show this: tracing the value t_{CA} in $g^{zwrs(x_{u,p}+h(p)t_{CA_*}(t+h(p)))}$ from g^r and $g^{t_{CA}}$ intuitively breaks the decisional Diffie-Hellman problem.

7 Conclusion and Future Work

In this paper we have presented a novel protocol called RevocationMatching, and showed how with it, we can support revocation in each of the different versions of Secret Handshake known in literature, own-group membership, dynamic matching and dynamic controlled matching. In the study of the security of the protocol, we have discovered an interesting new complexity assumption; we plan to analyze in more details its relationship with other complexity assumptions as well as its possible use in signature schemes and OSBE schemes. Moreover, we intend to study more closely dynamic accumulators: although they appear not to be perfectly suited for symmetric handshakes, they represent an interesting alternative when revocation requirements clash with untraceable credentials.

References

1. G. Ateniese, M. Blanton, and J. Kirsch. Secret handshakes with dynamic and fuzzy matching. In *Network and Distributed System Security Symposuim*, pages 159–177. The Internet Society, 02 2007. CERIAS TR 2007-24.
2. G. Ateniese, J. Camenisch, S. Hohenberger, and B. de Medeiros. Practical group signatures without random oracles, 2005.
3. A. Bagherzandi, J.-H. Cheon, and S. Jarecki. Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma. In *CCS '08: Proceedings of the 15th ACM conference on Computer and communications security*, pages 449–458, New York, NY, USA, 2008. ACM.
4. R. W. Baldwin and W. C. Gramlich. Cryptographic protocol for trustable match making. *Security and Privacy, IEEE Symposium on*, 1985.
5. D. Balfanz, G. Durfee, N. Shankar, D. K. Smetters, J. Staddon, and H.-C. Wong. Secret handshakes from pairing-based key agreements. In *IEEE Symposium on Security and Privacy*, pages 180–196, 2003.
6. J. Benaloh and G. Automation. One-way accumulators: A decentralized alternative to digital signatures. pages 274–285. Springer-Verlag, 1993.
7. D. Boneh and X. Boyen. Efficient selective-id secure identity-based encryption without random oracles. In *EUROCRYPT*, pages 223–238, 2004.
8. D. Boneh and X. Boyen. Short signatures without random oracles and the sdh assumption in bilinear groups. *J. Cryptology*, 21(2):149–177, 2008.

9. J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *Proceedings of Crypto 2002, volume 2442 of LNCS*, pages 61–76. Springer-Verlag, 2002.
10. C. Castelluccia, S. Jarecki, and G. Tsudik. Secret handshakes from ca-oblivious encryption. In *ASIACRYPT*, pages 293–307, 2004.
11. J. W. Changshe Ma and D. Zheng. Fast digital signature schemes as secure as diffie-hellman assumptions. Cryptology ePrint Archive, Report 2007/019, 2007.
12. J.-H. Hoepman. Private handshakes. In F. Stajano, C. Meadows, S. Capkun, and T. Moore, editors, *ESAS*, volume 4572 of *Lecture Notes in Computer Science*. Springer, 2007.
13. G. Jain. Zero knowledge proofs: A survey. 2008.
14. S. Jarecki, J. Kim, and G. Tsudik. Beyond secret handshakes: Affiliation-hiding authenticated key exchange. In *CT-RSA*, pages 352–369, 2008.
15. S. Jarecki and X. Liu. Unlinkable secret handshakes and key-private group key management schemes. In *ACNS '07: Proceedings of the 5th international conference on Applied Cryptography and Network Security*, pages 270–287, Berlin, Heidelberg, 2007. Springer-Verlag.
16. N. Li, W. Du, and D. Boneh. Oblivious signature-based envelope. *Distrib. Comput.*, 17(4):293–302, 2005.
17. A. Lysyanskaya, R. L. Rivest, A. Sahai, and S. Wolf. Pseudonym systems. In *Selected Areas in Cryptography*, pages 184–199, 1999.
18. C. Meadows. A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. *sp*, 0:134, 1986.
19. S. Nasserian and G. Tsudik. Revisiting oblivious signaturebased envelopes: New constructs and properties. In *In Financial Cryptography and Data Security (FC06*, 2006.
20. B. Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. pages 480–494. Springer-Verlag, 1997.
21. J. S. Shin and V. D. Gligor. A new privacy-enhanced matchmaking protocol. In *Network and Distributed System Security Symposuim*. The Internet Society, 02 2007.
22. V. Shoup. Lower bounds for discrete logarithms and related problems. In *EUROCRYPT*, pages 256–266, 1997.
23. A. Sorniotti and R. Molva. A provably secure secret handshake with dynamic controlled matching. In *Proceedings of The 24th International Information IFIP SEC 2009, May 18-20, 2009, Paphos, Cyprus*, 2009.
24. R. Tso, C. Gu, T. Okamoto, and E. Okamoto. Efficient id-based digital signatures with message recovery. In *CANS*, pages 47–59, 2007.
25. D. Vergnaud. Rsa-based secret handshakes. In *WCC*, pages 252–274, 2005.
26. B. Waters. Efficient identity-based encryption without random oracles. In *EUROCRYPT*, pages 114–127, 2005.
27. S. Xu and M. Yung. k-anonymous secret handshakes with reusable credentials. In *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security.*
28. K. Zhang and R. Needham. A private matchmaking protocol, 2001.

A Security of the New Assumption in Generic Groups

In what follows we will provide evidence as to the hardness of the problem introduced in Definitions 3, by proving a lower bound on the computational complexity under the generic group model. The generic group model is a theoretical framework for the analysis of the success of algorithms in groups where the representation of the elements reveals no information to the attacker. The most popular is the one presented by Victor Shoup [22]. In this model, the attacker is not given direct access to group elements, but rather to their images of group elements under a random one-to-one mapping. The only operations the attacker can perform are therefore equality testing by a bit-wise comparison on the images. Group operations can be computed by the attacker through a series of oracles. It is clear that in this situation, the attacker can gain no advantage in solving a computational problem from the representation of the group element. The model has been used to provide evidence as to the hardness of several computational problems [2, 8, 17].

Internally, the simulator represents the elements of $\mathbb{G}_1$ as their discrete logarithms relative to a chosen generator. To represent the images of the elements of $\mathbb{G}_1$ for the attacker, we use a random one-to-one mapping $\xi_1 : \mathbb{Z}_q^* \rightarrow \{0,1\}^{\lceil log_2 q \rceil}$, where q is the group order. For instance, the group element g^a is represented internally as a, whereas the attacker is given the external string representation $\xi_1(a) \in \{0,1\}^{\lceil log_2 q \rceil}$. We similarly define a second mapping $\xi_2 : \mathbb{Z}_q^* \rightarrow \{0,1\}^{\lceil log_2 q \rceil}$ to represent $\mathbb{G}_2$, and a third mapping $\xi_T :\rightarrow \{0,1\}^{\lceil log_2 q \rceil}$ to represent

$\mathbb{G}_T$. The adversary communicates with the oracles using the string representation of the group elements exclusively. Notice that the adversary is given $q = |\mathbb{G}_1| = |\mathbb{G}_2| = |\mathbb{G}_T|$.

The following theorem establishes the unconditional hardness of the SM problem in the generic bilinear group model. Our proof uses a technique similar to the one adopted by Ateniese et al. in [2].

Theorem 1. *Suppose $\mathcal{A}$ is an algorithm that is able to solve the SMproblem in generic bilinear groups of order q, making at most q_G oracle queries for the group operations in $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$, the oracle $O_{w,y}(\cdot)$ and the bilinear pairing $\hat{e}$, all counted together. Suppose also that the integers $w, y, m \in \mathbb{Z}_q^*$ and the encoding functions ξ_1, ξ_2, ξ_T are chosen at random. Then, the probability ϵ that $\mathcal{A}$ on input $(q, \xi_1(1), \xi_1(w), \xi_2(1), \xi_2(y), \xi_2(m))$ produces in output $(\xi_1(r), \xi_1(rs(x_* + y)), \xi_2((sw)^{-1}), \xi_2((s)^{-1}), \xi_T(rx_*m))$ with x_* not previously queried to $O_{w,y}$, is bounded by*

$$\epsilon \leq \frac{(q_G + 5)^2}{q} = O(q_G^2/q)$$

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

$\mathcal{B}$ maintains three lists of pairs $L_1 = \{(F_{1,i}, \xi_{1,i}) : i = 1, \ldots, \tau_1\}$, $L_2 = \{(F_{2,i}, \xi_{2,i}) : i = 1, \ldots, \tau_2\}$ and $L_T = \{(F_{T,i}, \xi_{T,i}) : i = 1, \ldots, \tau_T\}$, such that, at step τ in the game, $\tau_1 + \tau_2 + \tau_T = \tau + 5$. The entries $F_{1,i}$, $F_{2,i}$ and $F_{T,i}$ are polynomials with coefficients in $\mathbb{Z}_q^*$. The entries $\xi_{1,i}$, $\xi_{2,i}$, $\xi_{T,i}$ will be all the strings given out to the adversary.

The lists are initialized at step $\tau = 0$ by setting $\tau_1 = 2$, $\tau_2 = 3$, $\tau_T = 0$ and assigning $F_{1,1} = 1$, $F_{1,2} = W$, $F_{2,1} = 1$, $F_{2,2} = Y$ and $F_{2,3} = M$ where W, Y and M are indeterminants. The corresponding $\xi_{1,\cdot}$ and $\xi_{2,\cdot}$ are set to random distinct strings. In what follows we describe how $\mathcal{B}$ answers $\mathcal{A}$'s query:

Group operations : $\mathcal{A}$ may request a group operation in $\mathbb{G}_1$ as a multiplication or as a division. Before answering a $\mathbb{G}_1$ query, the simulator $\mathcal{B}$ starts by incrementing the τ_1 counter by one. $\mathcal{A}$ gives $\mathcal{B}$ two operands $\xi_{1,i}$, $\xi_{1,j}$ with $1 \leq i, j < \tau_1$, and a multiply/divide selection bit. To respond, $\mathcal{B}$ creates a polynomial $F_{1,\tau_1} \leftarrow F_{1,i} \pm F_{1,j}$. If the result is identical to an earlier polynomial $F_{1,l}$ for some $l < \tau_1$, the simulator $\mathcal{B}$ duplicates its string representation $\xi_{1,\tau_1} \leftarrow \xi_{1,l}$; otherwise, it lets ξ_{1,τ_1} be a fresh random string in $\{0,1\}^{\lceil log_2 q \rceil}$, distinct from $\xi_{1,1}, \ldots, \xi_{1,\tau_1 - 1}$. The simulator appends the pair $(F_{1,\tau_1}, \xi_{1,\tau_1})$ to the list $L - 1$ and gives the string ξ_{1,τ_1} back to $\mathcal{A}$. Group operation queries in $\mathbb{G}_2$ and $\mathbb{G}_T$ are answered in a similar way, based on the lists L_2 and L_T respectively.

Pairing : a pairing query consists of two operands $\xi_{1,i}$ and $\xi_{2,j}$ with $1 \leq i \leq \tau_1$ and $1 \leq j \leq \tau_2$ for the current values of τ_1 and τ_2. Upon receipt of such a query from $\mathcal{A}$, the counter τ_T is incremented. The simulator then computes the product of polynomials $F_{T,\tau_T} \leftarrow F_{1,i} \cdot F_{2,j}$. If the same polynomial was already present in L_T, i.e., if $F_{T,\tau_T} = F_{T,l}$ for some $l < \tau_T$, then $\mathcal{B}$ simply clones the associated string $\xi_{T,\tau_T} \leftarrow \xi_{T,l}$, otherwise it sets ξ_{T,τ_T} to a new random string in $\{0,1\}^{\lceil log_2 q \rceil}$, distinct from $\xi_{T,1}, \ldots, \xi_{1,\tau_T - 1}$. The simulator then adds the pair $(F_{T,\tau_T}, \xi_{T,\tau_T})$ to the list L_T, and gives the string ξ_{T,τ_T} to $\mathcal{A}$.

Oracle O : let τ_O be a counter initialized to 0 and $\mathcal{O}$ an empty set. At the beginning of any oracle query, $\mathcal{A}$ inputs $x \in \mathbb{Z}_q^*$; to start, $\mathcal{B}$ adds x to the set $\mathcal{O}$ and increments the counter τ_1 and τ_O by one, and the counter τ_2 by two, choosing a *new* indeterminant Z_{τ_O}; it then sets $F_{1,\tau_1} \leftarrow Z_{\tau_O}(x + Y)$; it also sets $F_{2,\tau_2 - 1} \leftarrow Z_{\tau_O}^{-1}$ and $F_{2,\tau_O} \leftarrow (Z_{\tau_O} W)^{-1}$. If the same polynomials were already present in L_1 or L_2, i.e., if $F_{1,\tau_1} = F_{1,l}$ for some $l < \tau_1$, or, for $j \in \{0,1\}$, $F_{2,\tau_2 - j} = F_{2,l'}$ for some $l' < \tau_2$, then $\mathcal{B}$ simply clones the associated string $\xi_{1,\tau_1} \leftarrow \xi_{T,l}$, $\xi_{2,\tau_2 - j} \leftarrow \xi_{2,l'}$; otherwise it sets the strings ξ_{1,τ_1} and $\xi_{2,\tau_2 - j}$ to distinct random values in $\{0,1\}^{\lceil log_2 q \rceil}$, different from the other strings already contained in the lists. The simulator then adds the pairs $(F_{1,\tau_1}, \xi_{1,\tau_1})$ to the list L_1 and $(F_{2,\tau_2 - j}, \xi_{2,\tau_2 - j})$ to the list L_2, giving the strings ξ_{1,τ_1} and $\xi_{2,\tau_2 - j}$ to $\mathcal{A}$.

We assume that the SXDH assumption holds, therefore we do not create any isomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$ or vice versa.

When $\mathcal{A}$ terminates, it returns the tuple $\langle \xi_{1,\alpha}, \xi_{1,\beta}, \xi_{2,\gamma}, \xi_{2,\delta}, \xi_{T,k} \rangle$ where $1 \leq \alpha, \beta, \leq \tau_1$, $1 \leq \gamma, \delta \leq \tau_2$ and $1 \leq k \leq \tau_T$. Let $F_{1,\alpha}$, $F_{1,\beta}$, $F_{2,\gamma}$, $F_{2,\delta}$ and $F_{T,k}$ be the corresponding polynomials in the lists L_1, L_2 and L_T, and g^{α}, g^{β}, $\tilde{g}^{\gamma}$, $\tilde{g}^{\delta}$, $\hat{e}(g, \tilde{g})^k$ the corresponding elements in $\mathbb{G}_1^2 \times \mathbb{G}_2^2 \times \mathbb{G}_T$.

In order to exhibit the correctness of $\mathcal{A}$'s answer, $\mathcal{B}$ should check that the system of equation

$$\begin{cases} \left(\dfrac{\hat{e}(g^{\beta}, \tilde{g}^{\gamma})}{\hat{e}(g^{\alpha}, \tilde{g}^{y})}\right)^{m} = \hat{e}\left(g, \tilde{g}\right)^{k} & (4) \\ \dfrac{\hat{e}\left(g, \tilde{g}^{\gamma}\right)}{\hat{e}\left(g^{w}, \tilde{g}^{\delta}\right)} = 1 & (5) \end{cases}$$

is verified. Let us set $\alpha = r$, $k = rx_*m$ and $\gamma = s^{-1}$, for some integers $r, x_*, s \in \mathbb{Z}_q^*$. If the above system is verified, we can rewrite $g^{\alpha} = g^{r}$, $g^{\beta} = g^{rs(x_*+y)}$, $\tilde{g}^{\gamma} = \tilde{g}^{(s)^{-1}}$, $\tilde{g}^{\delta} = \tilde{g}^{(ws)^{-1}}$, $\hat{e}\left(g, \tilde{g}\right)^{k} = \hat{e}\left(g, \tilde{g}\right)^{rx_*m}$; if $x_* \notin \mathcal{O}$ the attacker has produced a valid answer, according to Definition 3.

In order to verify the system above within the simulation framework, $\mathcal{B}$ computes

$$\begin{cases} F_{T,*} = \left(F_{1,\beta} \cdot F_{2,\gamma} - F_{1,\alpha} \cdot Y\right) M - F_{T,K} & (6) \\ F_{T,\circ} = F_{2,\gamma} - F_{2,\delta} \cdot W & (7) \end{cases}$$

To proceed with our demonstration, first of all we show that it is not possible that $F_{T,*} = F_{T,\circ} = 0$ for every value of W, Y, M and Z_i, $1 \le i \le \tau_O$. This result implies that the success of $\mathcal{A}$ in the game must depend on the particular values assigned to W, Y, M and Z_i.

Let us first observe that the polynomials $F_{1,\alpha}$, $F_{1,\beta}$ are by construction formed as

$$F_{1,\alpha} = \alpha_0 + \alpha_1 W + \sum_{i=1}^{\tau_O}(\alpha_{2,i} Z_i(x_i + Y))$$

$$F_{1,\beta} = \beta_0 + \beta_1 W + \sum_{i=1}^{\tau_O}(\beta_{2,i} Z_i(x_i + Y))$$

where x_i is the element of $\mathbb{Z}_q^*$ queried upon the i-th query to the oracle O. The polynomials $F_{2,\gamma}$ and $F_{2,\delta}$ instead are formed as

$$F_{2,\gamma} = \gamma_0 + \gamma_1 Y + \gamma_2 M + \sum_{i=1}^{\tau_O}(\gamma_{3,i} Z_i^{-1} + \gamma_{4,i}(Z_i W)^{-1})$$

$$F_{2,\delta} = \delta_0 + \delta_1 Y + \delta_2 M + \sum_{i=1}^{\tau_O}(\delta_{3,i} Z_i^{-1} + \delta_{4,i}(Z_i W)^{-1})$$

Plugging these equations back in Equation 7, gives us

$$\gamma_0 + \gamma_1 Y + \gamma_2 M + \sum_{i=1}^{\tau_O}(\gamma_{3,i} Z_i^{-1} + \gamma_{4,i}(Z_i W)^{-1}) =$$

$$\delta_0 W + \delta_1 WY + \delta_2 WM + \sum_{i=1}^{\tau_O}(\delta_{3,i} Z_i^{-1} W + \delta_{4,i} Z_i^{-1}) \qquad (8)$$

If the attacker wins the game, Equation 8 must be symbolically equal to zero; simplifying all the unique terms, we are left with

$$\sum_{i=1}^{\tau_O}(\gamma_{3,i} Z_i^{-1}) = W \sum_{i=1}^{\tau_O}(\delta_{4,i}(Z_i W)^{-1}) \qquad (9)$$

from which we conclude that $F_{2,\gamma} = \sum_{i=1}^{\tau_O}(\gamma_{3,i} Z_i^{-1})$.

Let us now consider Equation 6, which can be rewritten as

$$\left(\left(\sum_{i=1}^{\tau_O}(\beta_0\gamma_{3,i}Z_i^{-1}+\beta_1\gamma_{3,i}Z_i^{-1}W+\sum_{j=1}^{\tau_O}(\beta_{2,j}\gamma_{3,i}Z_i^{-1}Z_j(x_j+Y)))\right)-\left(\alpha_0Y+\alpha_1WY+\sum_{i=1}^{\tau_O}(\alpha_{2,i}Z_iY(x_i+Y))\right)\right)M=F_{T,K} \tag{10}$$

If the attacker wins the game, Equation 10 must be symbolically equal to zero. First of all, we notice that each term of the left hand of the equation contains M. Therefore, from $F_{T,K}$ we delete all the terms that do not contain M. Then, we simplify M on both sides. Further more, we simplify all the unique terms, ending up with

$$\beta_{2,j}\gamma_{3,i}(x_j+Y)-\alpha_0Y=K_0 \tag{11}$$

Now, $\alpha_0=\beta_{2,j}\gamma_{3,i}$ since they are the only coefficients of Y. Then $K_0=\beta_{2,j}\gamma_{3,i}x_j$. However this is not a valid solution, x_j is the j-th value queried to oracle O, and thus belongs to $\mathcal{O}$. We therefore conclude that it is impossible for the attacker to win the game for every value of W, Y, M and Z_i; instead this depends on a lucky instantiation of such variables.

The simulator $\mathcal{B}$ therefore chooses random values $\bar{w},\bar{y},\bar{m},\bar{z}_1,\ldots,\bar{z}_{\tau_O}$ for each of the variables W, Y, M and Z_i. Let us analyze the probability that the attacker has won the game given the chosen assignment of the variables: this happens if (i) no two non-identical polynomials in the lists L_1, L_2 and L_T assume the same value and (ii) if the assignment satisfies $F_{T,*}=F_{T,\circ}=0$. If (i) is true, $\mathcal{B}$'s simulation was flawed because two group elements – that were equal – have been presented as distinct to the attacker.

Summing up, the probability of success of the attacker is bounded by the probability that any of the following equations holds:

$$F_{1,i}(\bar{w},\bar{y},\bar{m},\bar{z}_1,\ldots,\bar{z}_{\tau_O})-F_{1,j}(\bar{w},\bar{y},\bar{m},\bar{z}_1,\ldots,\bar{z}_{\tau_O})=0,\quad i,j\text{ s.t. }F_{1,i}\neq F_{1,j} \tag{12}$$
$$F_{2,i}(\bar{w},\bar{y},\bar{m},\bar{z}_1,\ldots,\bar{z}_{\tau_O})-F_{2,j}(\bar{w},\bar{y},\bar{m},\bar{z}_1,\ldots,\bar{z}_{\tau_O})=0,\quad i,j\text{ s.t. }F_{2,i}\neq F_{2,j} \tag{13}$$
$$F_{T,i}(\bar{w},\bar{y},\bar{m},\bar{z}_1,\ldots,\bar{z}_{\tau_O})-F_{T,j}(\bar{w},\bar{y},\bar{m},\bar{z}_1,\ldots,\bar{z}_{\tau_O})=0,\quad i,j\text{ s.t. }F_{T,i}\neq F_{T,j} \tag{14}$$
$$F_{T,*}(\bar{w},\bar{y},\bar{m},\bar{z}_1,\ldots,\bar{z}_{\tau_O})-1=0 \tag{15}$$
$$F_{T,\circ}(\bar{w},\bar{y},\bar{m},\ldots,\bar{z}_{\tau_O})-1=0 \tag{16}$$

For fixed i,j each non-trivial polynomial 12, 13, 14 has degree at most 1 and it vanishes with probability $\leq 1/q$. Polynomials 15 and 16 have too degree at most 1 and vanish with probability $\leq 1/q$. We sum over all the (i,j) to bound the overall success probability ϵ of the attacker $\mathcal{A}$ as $\epsilon\leq\binom{\tau_1}{2}\frac{1}{q}+\binom{\tau_2}{2}\frac{1}{q}+\binom{\tau_T}{2}\frac{1}{q}+\frac{2}{q}$. Since $\tau_1+\tau_2+\tau_T\leq q_G+5$, we end up with

$$\epsilon\leq\frac{(q_G+5)^2}{q}=O(q_G^2/q)$$

□

B Proofs of security of RevocationMatching

In this section we present proofs of the security claims presented in Section 6.1.

B.1 Proof of Lemma 1

Proof. We define $\mathcal{B}$ as follows. $\mathcal{B}$ is given an instance $\langle g,g^a,g^b,g^\sigma\rangle$ of the DDH problem in $\mathbb{G}_1$ and wishes to use $\mathcal{A}$ to decide if $\sigma=ab$. The algorithm $\mathcal{B}$ simulates an environment in which $\mathcal{A}$ operates, using $\mathcal{A}$'s advantage in the game TraceProperty to help compute the solution to the DDH problem.

Setup Here is a high-level description of how the algorithm $\mathcal{B}$ will work. $\mathcal{B}$ uses g as the one received from the DDH challenge, picks and publishes the public parameters according to the rules of the protocol.

Queries At first, $\mathcal{A}$ queries $\mathcal{B}$ for an arbitrary number of tuples $\langle cred_{u_i,p_i}, match_{p_i},\ \ x_{u_i,p_i}, rev_{u_i,p_i}\rangle$ for any given pairs $(u_i, p_i) \in \mathcal{U} \times \mathcal{P}$. The queries can be adaptive. $\mathcal{B}$ answers truthfully abiding by the rules of the protocol.

Challenge At the end of this phase $\mathcal{A}$ initiates two handshake instances by sending e^{m_1} and e^{m_2}; $\mathcal{B}$ picks $x_1, x_2, s_1, s_2, r \overset{R}{\leftarrow} \mathbb{Z}_q^*$ and $p \overset{R}{\leftarrow} \mathcal{P}$ and generates two handshake tuples as follows:

$$\left\langle g^r, g^{rs_1(x_1+h(p)(bt+h(p)))}, \tilde{g}^{(s_1)^{-1}}, \tilde{g}^{(s_1 w)^{-1}} \right\rangle$$
$$\left\langle g^a, g^{as_2(x_2+h^2(p))} g^{\sigma s_2 h(p) t}, \tilde{g}^{(s_2)^{-1}}, \tilde{g}^{(s_2 w)^{-1}} \right\rangle$$

Analysis of $\mathcal{A}$'s response It is straightforward to verify that, if $\mathcal{A}$ wins the game, $\mathcal{B}$ can give the same answer to solve the DDH problem. Indeed, if $\mathcal{A}$ wins the game, she is able to decide if $\exists \alpha \in \mathbb{Z}_q^*$ such that

$$\begin{cases} (r(x_1 + h(p)(bt + h(p))) - r\alpha)m_1 = rm_1x_1 \\ (a(x_2 + h^2(p)) + \sigma h(p)t - a\alpha)m_2 = am_2x_2 \end{cases} \tag{17}$$

If $\mathcal{A}$'s answer is positive, it means that the system of equations is verified. Then we can solve the first equation as $\alpha = h(p)bt + h^2(p)$, and plugging in the second equation $\mathcal{B}$ can verify that $\sigma = ab$, which is the positive answer to the DDH problem. If not, $\mathcal{B}$ can give the negative answer to DDH. □

B.2 Proof of Lemma 2

Proof. We define $\mathcal{B}$ as follows. $\mathcal{B}$ is given an instance $\langle g, g^a, g^b, g^\sigma \rangle$ of the DDH problem in $\mathbb{G}_1$ and wishes to use $\mathcal{A}$ to decide if $\sigma = ab$. The algorithm $\mathcal{B}$ simulates an environment in which $\mathcal{A}$ operates, using $\mathcal{A}$'s advantage in the game TraceCredential to help compute the solution to the DDH problem.

Setup Here is a high-level description of how the algorithm $\mathcal{B}$ will work. $\mathcal{B}$ uses g as the one received from the DDH challenge, picks and publishes the public parameters according to the rules of the protocol.

Queries $\mathcal{A}$ can query $\mathcal{B}$ for an arbitrary number of tuples $\langle cred_{u_i,p_i}, match_{p_i},\ \ x_{u_i,p_i}, rev_{u_i,p_i}\rangle$ for any given pairs $(u_i, p_i) \in \mathcal{U} \times \mathcal{P}$. The queries can be adaptive. $\mathcal{B}$ answers truthfully abiding by the rules of the protocol.

Challenge At the end of this phase, $\mathcal{A}$ chooses a property p_*; $\mathcal{B}$ picks $r, s_1, s_2 \overset{R}{\leftarrow} \mathbb{Z}_q^*$ and prepares two handshake tuples as follows:

$$\left\langle g^r, g^{rs_1(b+h(p)(t+h(p)))}, \tilde{g}^{(s_1)^{-1}}, \tilde{g}^{(s_1 w)^{-1}} \right\rangle$$
$$\left\langle g^a, g^{s_2\sigma} g^{as_2(h(p)(t+h(p)))}, \tilde{g}^{(s_2)^{-1}}, \tilde{g}^{(s_2 w)^{-1}} \right\rangle$$

Analysis of $\mathcal{A}$'s response It is straightforward to verify that, if $\mathcal{A}$ wins the game, $\mathcal{B}$ can give the same answer to solve the DDH problem. Indeed, if $\mathcal{A}$ wins the game, she is able to tell if both handshake messages contain the same identification handle x_*. Let us assume this is the case. Then, the same revocation handle $rev_* = \tilde{g}^{x_*}$ can be used to revoke both credentials. Then, performing a check as described in Equation 3, the following system

$$\begin{cases} r(b + h(p)(t + h(p))) - rh(p)(t + h(p)) = rx_* \\ \sigma + a(h(p)(t + h(p))) - ah(p)(t + h(p)) = ax_* \end{cases} \tag{18}$$

should hold.

Then we can solve the first equation as $x_* = b$, and plugging in the second equation $\mathcal{B}$ can verify that $\sigma = ab$, which is the positive answer to the DDH problem. If not, $\mathcal{B}$ can give the negative answer to DDH. □

B.3 Proof of Lemma 3

Proof. We define $\mathcal{B}$ as follows. $\mathcal{B}$ is given an instance $\langle g, g^w, \tilde{g}, \tilde{g}^y, \tilde{g}^m, O_{w,y}\rangle$ of the SM problem and wishes to use $\mathcal{A}$ to produce the tuple $\langle g^r,\ g^{rs(x_*+y)},\ \tilde{g}^{s^{-1}}, \tilde{g}^{(ws)^{-1}}, \hat{e}\left(g, \tilde{g}\right)^{rx_*m}\rangle$, such that x_* has not been queried to O. The algorithm $\mathcal{B}$ simulates an environment in which $\mathcal{A}$ operates.

Setup Here is a high-level description of how the algorithm $\mathcal{B}$ will work. $\mathcal{B}$ sets public parameters $g, \tilde{g}$ as the ones received from the challenge. It then sets $W \leftarrow g^w$, $T \leftarrow \tilde{g}^y$; the other public parameters are set according to the rules of the protocol.

Queries $\mathcal{A}$ queries $\mathcal{B}$ for an arbitrary number of tuples $\langle cred_{u_i,p_i}, match_{p_i},\ x_{u_i,p_i}, rev_{u_i,p_i}\rangle$ for any given pairs $(u_i, p_i) \in \mathcal{U}\times\mathcal{P}$. The queries can be adaptive. Upon a query for (u_i, p_i), $\mathcal{B}$ answers picking $x_{u_i,p_i} \xleftarrow{R} \mathbb{Z}_q^*$; $\mathcal{B}$ then queries the oracle $O_{w,y}$ providing $\frac{x_{u_i,p_i}+h^2(p)}{h(p)}$ as input, adding the value $\frac{x_{u_i,p_i}+h^2(p)}{h(p)}$ to the set $\mathcal{O}$ of queries to oracle O. The output of the oracle is $(g^{z(\frac{x_{u_i,p_i}+h^2(p)}{h(p)}+y)}, \tilde{g}^{z^{-1}}, \tilde{g}^{(zw)^{-1}})$. $\mathcal{B}$ then assigns $C_{u_i,p_i,1} \leftarrow \left(g^{z(\frac{x_{u_i,p_i}+h^2(p)}{h(p)}+y)}\right)^{h(p)}$, $C_{u_i,p_i,2} \leftarrow \tilde{g}^{z^{-1}}$, $C_{u_i,p_i,3} \leftarrow \tilde{g}^{(zw)^{-1}}$, $match_{p_i} = \left(T\tilde{H}(p)\right)^{h(p)}$, $rev_{u_i,p_i} = \tilde{g}^{x_{u_i,p_i}}$ and gives the requested parameters to $\mathcal{A}$. The attacker can successfully perform all the checks mandated by the protocol; his view is therefore undistinguishable from a standard protocol instantiation.

Challenge $\mathcal{A}$ then declares that this phase of the game is over. $\mathcal{B}$ therefore revokes each of the credentials $\mathcal{A}$ requested in the previous phase. $\mathcal{A}$ then chooses a property $p_* \in \mathcal{P}$. $\mathcal{A}$ receives from $\mathcal{B}$ the matching reference $\left(T\tilde{H}(p)\right)^{h(p)}$ of property p_*. $\mathcal{B}$ challenges $\mathcal{A}$ by sending $\hat{e}\left(g, \tilde{g}^m\right)$ and $\mathcal{A}$ answers the challenge with the tuple $\langle g^\alpha, g^\beta, \tilde{g}^\gamma, \tilde{g}^\delta,\ e^k\rangle$.

Analysis of $\mathcal{A}$'s response If $\mathcal{A}$ wins the game, $\mathcal{B}$ can check that

$$\left(\frac{\hat{e}(g^\beta, \tilde{g}^\gamma)}{\hat{e}(g^\alpha, match_{p_*})}\right)^m = \left(\frac{\hat{e}(g^\beta, \tilde{g}^\gamma)}{\hat{e}\left(g^\alpha, \tilde{g}^{h(p_*)(y+h(p_*))}\right)}\right)^m = e^k \tag{19}$$

and that

$$\hat{e}\left(g^w, \tilde{g}^\delta\right) = \hat{e}\left(g, \tilde{g}^\gamma\right) \tag{20}$$

as mandated by the $\mathsf{Authenticate}$ step of $\mathsf{RevocationMatching}$ described in Section 4.

Let us set $\alpha = r$, $k = rx_*m$ and $\gamma = s^{-1}$, for some integers $r, x_*, s \in \mathbb{Z}_q^*$ unknown to $\mathcal{B}$. Then, from Equation 20 we derive that $\delta = (ws)^{-1}$ and from Equation 19 that $\beta = rs(x_* + h(p_*)(y + h(p_*)))$. Notice that by the definition of the game, the attacker has not received a credential containing $x_* + h(p_*)(y + h(p_*))$; factoring $h(p_*)$ we derive that $\frac{x_*+h^2(p_*)}{h(p_*)}$ cannot belong to the set $\mathcal{O}$. Therefore we conclude that, if $\mathcal{A}$ wins the game, $\mathcal{B}$ can provide

$$\left\langle \left(g^\alpha\right)^{h(p_*)}, g^\beta, \tilde{g}^\gamma, \tilde{g}^\delta, \hat{e}\left(g, \tilde{g}\right)^k \cdot \hat{e}\left(g^\alpha, \tilde{g}^m\right)^{h^2(p_*)}\right\rangle$$

as an answer to the SM problem. □

B.4 Proof of Lemma 4

Proof. We define $\mathcal{B}$ as follows. $\mathcal{B}$ is given an instance $\left\langle g, g^a, g^b, g^c, \tilde{g}, \tilde{g}^a, \tilde{g}^b, \tilde{g}^\sigma\right\rangle$ of the BDDH problem and wishes to use $\mathcal{A}$ to decide if $\sigma = abc$. The algorithm $\mathcal{B}$ simulates an environment in which $\mathcal{A}$ operates.

Setup Here is a high-level description of how the algorithm $\mathcal{B}$ will work. $\mathcal{B}$ sets $g, \tilde{g}$ as the ones received from the BDDH instance; T is set to be equal to $\tilde{g}^{at}$. It then sets all the remaining parameters as mandated in the rules of the protocol.

Queries $\mathcal{A}$ queries $\mathcal{B}$ for an arbitrary number of $\langle cred_{u_i,p_i}, match_{p_i},\ x_{u_i,p_i}, rev_{u_i,p_i}\rangle$ for any given pairs $(u_i, p_i) \in \mathcal{U} \times \mathcal{P}$. The queries can be adaptive. $\mathcal{B}$ answers by picking for each query $x_{u_i,p_i}, z \xleftarrow{R} \mathbb{Z}_q^*$, and giving $C_{u_i,p_i,1} \leftarrow g^{z(x_{u_i,p_i}+h(p_i)(at+h(p_i)))}$, $C_{u_i,p_i,2} = \tilde{g}^{z^{-1}}$, $C_{u_i,p_i,3} = \tilde{g}^{(zw)^{-1}}$, $match_{p_i} = \tilde{g}^{h(p_i)(at+h(p_i))}$, $rev_{u_i,p_i} = \tilde{g}^{x_{u_i,p_i}}$ to the attacker. The attacker can successfully perform all the checks mandated by the protocol; his view is therefore undistinguishable from a standard protocol instantiation. $\mathcal{B}$ adds to a list V the tuple $(\tilde{g}^{x_{u_i,p_i}+h(p_i)(at+h(p_i))}, u_i, p_i, x_{u_i,p_i})$ for each query of $\mathcal{A}$ and keeps it for later use.

Challenge $\mathcal{A}$ then declares that this phase of the game is over. $\mathcal{B}$ therefore revokes each credential requested by $\mathcal{A}$ in the previous phase. $\mathcal{A}$ then declares property $p_* \in \mathcal{P}$. $\mathcal{B}$ challenges $\mathcal{A}$ by sending $\hat{e}\left(g, \tilde{g}\right)^{bc}$ and $\mathcal{A}$ answers the challenge with the tuple $\langle g^\alpha, g^\beta, \tilde{g}^\gamma, \tilde{g}^\delta, \hat{e}\left(g, \tilde{g}\right)^k\rangle$.

Analysis of $\mathcal{A}$'s response If $\mathcal{A}$ wins the game, $\mathcal{B}$ can check that

$$\left(\frac{\hat{e}(g^\beta, \tilde{g}^\gamma)}{\hat{e}(g^\alpha, match_{p_*})}\right)^{bc} = \hat{e}\left(g, \tilde{g}\right)^k \tag{21}$$

and that

$$\hat{e}\left(g^w, \tilde{g}^\delta\right) = \hat{e}\left(g, \tilde{g}^\gamma\right) \tag{22}$$

as mandated by the Authenticate step of RevocationMatching described in Section 4.

Let us set $\alpha = r$, $k = rx_*bc$, $\gamma = s^{-1}$ and $\beta = rsv_*$ for some integers $r, x_*, s, v_* \in \mathbb{Z}_q^*$ unknown to $\mathcal{B}$. Then, from Equation 22 we derive that $\delta = (ws)^{-1}$.

We know by definition that the attacker has already received $C_{u_\circ,p_\circ,1} = g^{zv_*} = g^{z(x_{u_\circ,p_\circ}+h(p_\circ)(at+h(p_\circ)))}$ during the previous query phase. Consequently, the revocation handle $rev_{u_\circ,p_\circ} = \tilde{g}^{x_{u_\circ,p_\circ}}$ has also been published. $\mathcal{B}$ can easily recover $u_\circ$ and $p_\circ$, since she can check for which $\tilde{g}^{x_{u_\circ,p_\circ}+h(p_\circ)(at+h(p_\circ))}$ in the list V, $\hat{e}(g^\beta, \tilde{g}^\gamma) = \hat{e}(g^\alpha, \tilde{g}^{x_{u_\circ,p_\circ}+h(p_\circ)(at+h(p_\circ))})$ holds and look up the respective $h(p_\circ)$ and $x_{u_\circ,p_\circ}$.

If $p_\circ = p_*$, then $\mathcal{A}$ has lost the game, since a successful answer of the attacker cannot be revoked by any of the issued revocation handles, whereas if $p_\circ = p_*$, the credential can be revoked with $rev_{u_\circ,p_\circ}$. Then it must be that $p_\circ \neq p_*$; in this case $x_* = x_{u_\circ,p_\circ} + h(p_\circ)(at + h(p_\circ)) - h(p_*)(at + h(p_*))$.

Then $k = rbc(x_{u_\circ,p_\circ} + h^2(p_\circ) - h^2(p_*)) + rabct(h(p_\circ) - h(p_*))$. However $\mathcal{B}$ is still not able to use $\mathcal{A}$'s answer to solve the BDDH problem since $\mathcal{B}$ cannot compute $\hat{e}\left(g, \tilde{g}\right)^{rbc(x_{u_\circ,p_\circ}+h^2(p_\circ)-h^2(p_*))}$. Using the generalized forking lemma [3] presented by Bagherzandi et al. and used to a similar end in [24, 11], we know that $\mathcal{A}$ can be executed twice with the same random tape that produced r but with different parameters. In particular, in the forked instance $\mathcal{B}$ can replace $T = \tilde{g}^{at}$ by $\tilde{g}^t$. Therefore from the forked instance, $\mathcal{B}$ recovers

$$\hat{e}\left(g, \tilde{g}\right)^{rbc} = \left(\hat{e}\left(g, \tilde{g}\right)^{k'}\right)^{(x_{u_\circ,p_\circ}+h(p_\circ)(t+h(p_\circ))-h(p_*)(t+h(p_*)))^{-1}}$$

and is finally able to decide if $\sigma = abc$ by checking if

$$\hat{e}\left(g^\alpha, \tilde{g}^\sigma\right) = \left(\frac{\hat{e}\left(g, \tilde{g}\right)^k}{\left(\hat{e}\left(g, \tilde{g}\right)^{rbc}\right)^{(x_{u_\circ,p_\circ}+h^2(p_\circ)-h^2(p_*))}}\right)^{((h(p_\circ)-h(p_*))t)^{-1}}$$

□