Enhanced Privacy ID: A Direct Anonymous Attestation Scheme with Enhanced Revocation Capabilities[*]

Ernie Brickell
Intel Corporation
ernie.brickell@intel.com

Jiangtao Li
Intel Corporation
jiangtao.li@intel.com

August 17, 2007

Abstract

Direct Anonymous Attestation (DAA) is a scheme that enables the remote authentication of a Trusted Platform Module (TPM) while preserving the user's privacy. A TPM can prove to a remote party that it is a valid TPM without revealing its identity and without linkability. In the DAA scheme, a TPM can be revoked only if the DAA private key in the hardware has been extracted and published widely so that verifiers obtain the corrupted private key. If the unlinkability requirement is relaxed, a TPM suspected of being compromised can be revoked even if the private key is not known. However, with the full unlinkability requirement intact, if a TPM has been compromised but its private key has not been distributed to verifiers, the TPM cannot be revoked. Furthermore, a TPM cannot be revoked from the issuer, if the TPM is found to be compromised after the DAA issuing has occurred. In this paper, we present a new DAA scheme called Enhanced Privacy ID (EPID) scheme that addresses the above limitations. While still providing unlinkability, our scheme provides a method to revoke a TPM even if the TPM private key is unknown. This expanded revocation property makes the scheme useful for other applications such as for driver's license. Our EPID scheme is efficient and provably secure in the same security model as DAA, i.e. in the random oracle model under the strong RSA assumption and the decisional Diffie-Hellman assumption.

1 Introduction

Direct Anonymous Attestation (DAA) is a scheme developed by Brickell, Camenisch, and Chen [10] for remote authentication of a hardware module, called Trusted Platform Module (TPM), while preserving the privacy of the user of the platform that contains the module. The DAA scheme was adopted by the Trusted Computing Group (TCG) [38], an industry standardization body that aims to develop and promote an open industry standard for trusted computing hardware and software building blocks, and was included in TPM specification version 1.2 [37].

In DAA, there exists an issuer who creates a group public key. Later on, each TPM obtains a unique membership private key from the issuer. To authenticate as a group member, the TPM generates a signature using his membership private key such that the signature can be verified by a verifier using the group public key. One notable feature in DAA is its revocation mechanism. If a TPM was compromised and its membership private key has been extracted from the hardware

[*]A preliminary version of this paper will be presented at the 6th Workshop on Privacy in the Electronic Society (WPES), Alexandria, Virginia, October 2007.

device and exposed to public, the private key is placed in a revocation list. Later, when a verifier verifies a signature from a TPM, the verifier can locally check this signature against the revocation list. This revocation mechanism is also known as verifier-local revocation [5].

In the DAA scheme, there are two options for a balance between linkability and revocation. If the random base option is used, i.e. a different base used every time a DAA signature is performed, then any two signatures by a TPM are unlinkable, but revocation only works if the corrupted TPM's private key has been revealed to the public. If a TPM has been compromised but its private key has not been distributed to the verifiers (e.g., the corrupted membership private key is still under control by the adversary), the corrupted TPM cannot be revoked. If the named base option is used, i.e., a deterministic function of the name of the verifier is used as a base, then any two signatures produced by a TPM using the same base are linkable. Thus, if the verifier determines that a membership private key that was used in a signature has been compromised, that verifier can revoke that key locally, i.e., reject all future signatures generated by that key, without knowing the compromised membership private key. However, the verifier cannot tell a different verifier that uses a different named base to revoke that private key, because when a different name is used and the revoked key cannot be identified. Furthermore, the privacy advocates were not fond of the named base option, because the verifier could link the transactions.

In the DAA scheme, during the issuing of a DAA private key, the issuer obtains the identity of the TPM, but does not learn the DAA membership private key. If sometime after issuing, the issuer discovers that the TPM has been compromised, the issuer cannot revoke the DAA private key that has been issued to that compromised TPM.

In this paper, we develop a new scheme called Enhanced Privacy ID (EPID) that addresses the above limitations. EPID scheme can be seem as a new DAA scheme with enhanced revocation capabilities. We believe that with this enhanced revocation capability, the new scheme will have broader applicability beyond attestation and the TCG application. In an EPID scheme, there are four types of entities: an issuer, a revocation manager, users, and verifiers. The issuer could be the same entity as the revocation manager. An EPID scheme has the following four procedures:

Setup In this procedure, the issuer creates a group public key and a group issuing private key. The issuer publishes the group public key.

Join This is a protocol between the issuer and a user that results in the user becoming a new group member. At the end of this protocol, the user obtains a membership private key from the issuer.

Proof of Membership In this protocol, a prover interacts with a verifier to convince the verifier that he is a member of the group in good standing (i.e., without being revoked). It has the following steps: (1) the prover sends a request to the verifier, (2) the verifier responds with a message m, (3) the prover generates a signature on m based on his membership private key, and (4) the verifier verifies the signature using the group public key.

Revocation The revocation manager puts a group member into the revocation list. There are three types of revocations: (1) private-key based revocation in which the revocation manager revokes a user based on the user's membership private key, (2) signature based revocation in which the revocation manager revokes a user based on the signatures created by the user, and (3) issuer based revocation in which the revocation manager revokes a user based on the recommendation from the issuer.

In an EPID scheme, the signatures generated in the proof of membership protocol must be (1) unforgeable, i.e., only non-revoked group members are able to generate valid signatures, (2) anonymous, i.e., the verifier cannot identify the actual signer given a valid signature, and (3) unlinkable, i.e., it is computationally infeasible to determine whether two different signatures were computed by the same group member. We shall formalize these security properties in Section 2.

In this paper, we develop the new EPID scheme. Our EPID scheme builds on top of the DAA scheme [10] and applies Camenisch-Lysyanskaya (CL) signature scheme [14] and the related protocols as underlying building blocks. For private-key based revocation, we still use verifier-local revocation [10, 5], i.e., the revocation check is done only at the verifier's side. For the other two types of revocation, we develop proof of knowledge protocols for proving that a user's membership private key is not listed in the revocation list. Essentially, we give protocols for proving the inequality of multiple discrete logarithms. These proof of knowledge protocols may be of independent interest in other applications as well. Our construction of EPID is efficient and provably secure in the random oracle model under the strong RSA assumption and the decisional Diffie-Hellman assumption.

A possible alternative to handle revocation is to add traceability to the DAA scheme, as many group signature schemes do. That is, we give the revocation manager the ability to open a signature and identify the actual signer. To revoke a user based on his signature, the revocation manager first finds out the user's private key or his identity, then put the user into the revocation list. As in DAA scheme, EPID scheme chooses *not* to have traceability from the revocation manager because we want to provide maximum privacy for the users. Traceability provides the capability that a revocation manager can determine which user generates which signatures without any acknowledgement from the user that is being traced. This is not desirable from a privacy perspective. With EPID, if a user's membership private key has been revoked, i.e., placed in a revocation list, the user will know that he is revoked or being traced. If the user finds that he is not in the revocation list, then he is assured that nobody can trace him, including the issuer and the revocation manager. Observe that, if the revocation manager does not have traceability and the signature cannot be opened, revocation based on signature is a much more challenging problem.

1.1 Application of EPID

1.1.1 Trusted Computing and TPM

As the DAA scheme, EPID can be used in trusted computing for remote authentication of TPM, a hardware module integrated into a platform such as a laptop or a mobile device. Consider the following scenario. A user of a platform communicates with a verifier who wants to be assured that the platform of the user indeed contains a certified TPM. At the same time, the user wants his privacy protected, i.e., the verifier only learns that the user uses a TPM but not which particular one. Let the group be the set of all valid TPMs. To use EPID, each TPM obtains a membership private key from the issuer. Later, a TPM can conduct a proof of membership to a verifier without revealing its identity. As we explained, EPID has better revocation capabilities than the DAA scheme [10]. When a verifier suspects that a TPM has been compromised, but has not obtained the membership private key of the compromised TPM, the verifier can reject any further signatures from the suspected TPM using our new revocation method.

1.1.2 Driver's License and Identity Card

Various governments are considering including machine readable information on driver's licenses and identity cards. One proposal is to use machine readable technology on driver's licenses, that is, the machine readable portion (e.g., bar code or magnetic strip) of the driver's license is readable to anyone with a license reader. Unfortunately, such approach raises serious privacy concerns as personal information in the licenses can be easily gathered and is often sold without the owners consent which could potentially lead to identity theft. Another proposal is to encrypt the machine readable portion of the license. This poses significant key management challenges to assure that the decryption is only available to authorized parties.

We describe how EPID can be applied to a driver's license. Each license has an embedded smart card chip that can store and process information. A card reader will be used to communicate with the driver's license. A driver's license runs the join protocol when it is issued by the Department of Motor Vehicles. The driver's license could be used for various purposes without violating the users privacy. The smart card license would be able to prove to the reader that it was a valid license and that it was not revoked, suspended, reported lost, etc. This can be done using the proof of membership protocol so that the identity of the license is not revealed. Each state would have multiple license groups for the issuing of licenses. When a license is proving to a reader that it is valid, it will reveal which license group it is in, and that it is a valid license in good standing, but it will not reveal which license it is within that license group.

Using the EPID scheme, the proof that the license is valid and in good standing is unlinkable. This means that if the license is used at one restaurant to prove that the license is valid and issued to someone of legal age, and the license is used again at the same restaurant the next night, the restaurant would not be able to tell that it was the same license that was being presented. The restaurant would only know exactly what it is allowed to know, that the license is valid, and that the patron was of age.

Consider the example of using of the driver's license at an airport. The airport check has several requirements. First, the airport agent needs to be assured that the license is valid, and that the license was issued to the same person who is flying. Second, the agent would want to know that the license was not on a list of compromised or lost licenses. Third, the agent would also want to know if the license belonged to a person who was wanted by law enforcement. Here is one option for meeting these requirements. The agent could check the picture on the front of the license to determine if it was the person flying. The agent could compare the name printed on the airline ticket to the name printed on the front of the license. The agent would obtain the list of compromised or lost licenses, and also the list of persons wanted by law enforcement for the relevant group of licenses. The agent would give these lists to the license and the license would respond with a proof that it was valid and not on those lists. As described below, the license would first validate the authenticity of these revocation lists before responding.

1.2 Related Work

The EPID scheme in this paper shares some properties with group signatures [1, 4, 18, 25], DAA [10], identity escrow [30], and anonymous credential systems [12, 21]. In fact, our scheme draws on techniques that have been developed in these schemes, e.g., building blocks from the DAA scheme [10] and the group signature schemes [1, 12, 14]. The EPID scheme differs from the DAA scheme in that it adds additional revocation capabilities. Another work related to EPID is the pseudonym system of Brands [7]. Brands' system provides efficient techniques for proving relations among committed values. However, the credentials in that system are linkable for multiple display, whereas the signatures in the EPID scheme are unlinkable.

There have been several revocation methods proposed for group signatures, such as [8, 36, 13, 2, 5]. In Bresson and Stern's revocation method [8], when proving membership, a user proves that his membership private key does not appear in the revocation list. This feature is similar to ours, however, their scheme requires the traceability feature which ours does not. Song [36] proposed an alternative approach for revocation, but her result does not work for ordinary group signature schemes. Ateniese, Song, and Tsudik [2] modified Song's revocation approach and applied the revocation method to a group signature scheme [1]. Although their solution requires only constant computational time for the prover, their solution uses so-called double discrete logarithms and is rather expensive (about a factor of 100 for reasonable security parameters). Camenisch and Lysyanskaya [13] proposed a revocation mechanism using dynamic accumulators. Their scheme takes constant time in revocation check for both the prover and the verifier. However, their scheme requires every group member to update his membership private key each time a join or a revoke operation takes place. The unique property that EPID has that none of the above have, is the capability to revoke a membership private key that generated a signature, without being able to open the signature.

1.3 Organization of This Paper

Rest of this paper is organized as follows. We first give a precise definition of security model for the EPID scheme in Section 2. We then define our notations, present security assumptions, and briefly review some previously known cryptographic techniques in Section 3. We describe our EPID scheme in Section 4 and give intuition into construction of our scheme in Section 5. We prove that our EPID scheme is secure under the decisional Diffie-Hellman assumption and the strong RSA assumption in Section 6. In Section 7, we describe the usage of our EPID scheme with tamper resistant hardware and how to achieve better efficiency with hardware support. We conclude this paper in Section 8.

2 Security Model

This section provides the formal security model of EPID. As in DAA scheme [10] and anonymous credentials scheme [12], we use an ideal-system/real-system model to prove security of EPID based on security models for multi-party computation [19, 20] and reactive systems [32, 33].

2.1 Overview of Security Model

We summarize the basic ideas of the ideal-system/real-system model as follows. In the real system there are a number of players who run some cryptographic protocols with each other, an adversary $\mathcal{A}$ who controls a set of dishonest players, and an environment $\mathcal{E}$ that (1) provides the players with inputs and (2) arbitrarily interacts with $\mathcal{A}$. The environment provides the inputs to the honest players and receives their outputs, and interacts arbitrarily with $\mathcal{A}$. The dishonest players are fully controlled by $\mathcal{A}$, who monitors all the messages sent to the dishonest players and generates output messages for them. We assume that the adversary in our model is static, i.e., the set of corrupted players is fixed during the execution of the protocols.

In the ideal system, we have the same players. However, they do not run any cryptographic protocols but send all their inputs to and receive all their outputs from an ideal trusted third party $\mathcal{T}$. This party computes the output of the players from their inputs, i.e., applies the functionality that the cryptographic protocols are supposed to realize.

A cryptographic protocol is said to implement securely a functionality if for every adversary $\mathcal{A}$ and every environment $\mathcal{E}$ there exists a simulator $\mathcal{S}$ controlling the same players in the ideal system as $\mathcal{A}$ does in the real system such that the environment $\mathcal{E}$ can not distinguish whether it is run in the real system and interacts with $\mathcal{A}$ or whether it is run in the ideal system and interacts with $\mathcal{S}$.

2.2 Ideal System of EPID

We now specify the functionality of EPID. We have the following types of players: an issuer, a revocation manager, users, and verifiers. The set of users in the system may grow over time. The issuer is the entity that grants membership certificates for the users. The revocation manager is the entity that revokes membership certificates. Note that our model can be extended to support multiple issuers and revocation managers.

In the ideal system, the trusted third party $\mathcal{T}$ supports the following operations:

Setup: Each player indicates to $\mathcal{T}$ whether or not it is corrupted by the adversary.

Join: A user contacts $\mathcal{T}$ and requests to become a group member. $\mathcal{T}$ asks the issuer whether the user can become a member. If the issuer agrees and replies yes, $\mathcal{T}$ notifies the user that he has become a member.

Proof of Membership: A prover interacts with a verifier to prove that he is a group member in good standing. The prover first sends a request to $\mathcal{T}$ that he wants to contact the verifier. $\mathcal{T}$ informs the verifier that somebody wants to perform the proof of membership without revealing to the verifier who is the prover. The verifier chooses a message m and sends m to $\mathcal{T}$, who forwards m to the prover. If the prover is not a member, $\mathcal{T}$ aborts. Otherwise, $\mathcal{T}$ tells the prover whether he has been revoked and asks him whether to proceed. If the prover does not abort, $\mathcal{T}$ proceeds as follows.

- If the prover has been revoked, $\mathcal{T}$ lets the verifier know that a revoked member has signed the message m.
- Otherwise, $\mathcal{T}$ informs the verifier that m has been signed by a legitimate member.

Revocation: The revocation manager tells $\mathcal{T}$ to revoke a user. If the user is not a group member, $\mathcal{T}$ denies the request. Otherwise, $\mathcal{T}$ marks the user as revoked.

2.3 Discussions

Now we briefly discuss the properties of our model. Observe that the ideal system of EPID captures both unforgeability and anonymity. For unforgeability, a user who is not a group member or is a group member but has been revoked cannot succeed in proof of membership to any verifiers. For anonymity, the verifier cannot identify who is the prover in a proof of membership operation. Furthermore, for any two proof of membership operations that involve the same verifier, the verifier cannot tell whether the operations are initiated by the same prover or two different provers.

In the ideal system, revocation has no effects on old signatures but only causes verifiers to reject messages signed by a revoked member. In the EPID protocol, however, if a user's private key is exposed and the user is revoked, the signatures from this revoked user become linkable to an honest verifier. As a result, corrupted users who reveal their private keys and are revoked deliberately lose their privacy. As in the DAA scheme [10], for simplicity, we do not model this in the ideal system and thus an honest verifier in the real system will not consider this information.

In the EPID protocol, a prover can check whether he has been revoked from on the revocation list, before he signs a signature and sends it to the verifier. If the prover finds out that he has been revoked, he can choose to not proceed. This is properly modeled in the ideal system, where the revocation list is transparent to the users. In the ideal system, when $\mathcal{T}$ forwards the verifier's message m to the prover, $\mathcal{T}$ also informs the prover whether he has been revoked. The prover can choose to abort before $\mathcal{T}$ contacts the verifier again. Observe that, the proof of membership operation can be terminated either because $\mathcal{T}$ aborts (i.e., the prover is not a group member) or because the prover aborts (i.e., the prover has been revoked). The verifier cannot distinguish which one is the case when an early termination happens.

In the DAA scheme [10], a user is revoked only if he has been corrupted and his private key has been exposed. In other words, only users controlled by $\mathcal{A}$ can be revoked. Whereas in the EPID scheme, the revocation manager is flexible in revoking any user as long as the user is a valid group member. This is reflected in the ideal system that $\mathcal{T}$ only verifies the membership of the user before revoking him.

3 Background

3.1 Notations

In the rest of this paper, we use the following notations. We use $\{0,1\}^\ell$ to denote the set of all binary strings of length ℓ. We often switch between integers and their binary representations, e.g., we write $\{0,1\}^\ell$ for the set $[0, 2^\ell - 1]$ of integers. Let (K, K^{-1}) be a public-private key pair, we use $\{m\}_{K^{-1}}$ to denote a message m signed by the private key K^{-1}.

We say that $\mu(k)$ is a negligible function, if for every polynomial $p(k)$ and for all sufficiently large k, $\mu(k) < 1/p(k)$. If S is a probability space, then the probability assignment $x \leftarrow S$ means that an element x is chosen at random according to S. If S is a finite set, then $x \leftarrow S$ denotes that x is chosen uniformly from S. Let A be an algorithm, we use $y \leftarrow A(x)$ to denote that y is obtained by running A on input x. In case A is deterministic, then y is unique; if A is probabilistic, then y is a random variable. Let p be a predicate and $A_1, A_2, \ldots, A_n$ be n algorithms then $\Pr\left[\{x_i \leftarrow A_i(y_i)\}_{1\le i\le n} : p(x_1, \cdots, x_n)\right]$ denotes the probability that $p(x_1, \cdots, x_n)$ will be true after running sequentially algorithms $A_1, \ldots, A_n$ on inputs $y_1, \ldots, y_n$.

3.2 Cryptographic Assumptions

The security of our EPID scheme relies on the strong RSA assumption and the decisional Diffie-Hellman (DDH) assumption.

Assumption 1 (Strong RSA Assumption)**.** *The strong RSA assumption states that it is computationally infeasible, on input a random RSA modulus N and a random element $u \in \mathbb{Z}_N^*$, to compute values $e > 1$ and v such that $v^e \equiv u \pmod N$. In other words, for every probabilistic polynomial-time algorithm A,*

$$\Pr\left[N \leftarrow G(1^k), u \leftarrow \mathbb{Z}_N^*, (v, e) \leftarrow A(n, u) \;:\; v^e \equiv u \pmod N \;\wedge\; 1 < e < N\right] = \mu(k)$$

where $G(1^k)$ is an algorithm that generates a RSA modulus and $\mu(k)$ is a negligible function.

The tuple (N, u) generated as above is called an *instance* of the *flexible* RSA problem.

Assumption 2 (DDH Assumption)**.** *Let p be an ℓ_p-bit prime and q is an ℓ_q-bit prime such that $q | p-1$. Let $g \in \mathbb{Z}_p^*$ be a random element of order q. Then, for sufficiently large values of ℓ_p and ℓ_q, the distribution $\{(g, g^a, g^b, g^{ab})\}$ is computationally indistinguishable from the distribution $\{(g, g^a, g^b, g^c)\}$, where a, b, and c are random elements from $\mathbb{Z}_q$. It can be formally stated as, for every probabilistic polynomial-time algorithm A,*

$$\left|\Pr[A(p, q, g, g^a, g^b, g^{ab}) = 1] \;-\; \Pr[A(p, q, g, g^a, g^b, g^c) = 1]\right| = \mu(k)$$

where $\mu(k)$ is a negligible function and the probabilities are taken over the choice of (p, q, g) according to some generation function $G(1^k)$ and the random choice of a, b, and c in $\mathbb{Z}_q$.

3.3 Protocols for Proof of Knowledge

In our scheme we will use various protocols to prove knowledge of and relations among discrete logarithms. To describe these protocols, we use notation introduced by Camenisch and Stadler [18] for various proofs of knowledge of discrete logarithms and proofs of the validity of statements about discrete logarithms. For example,

$$PK\{(a, b) : y_1 = g_1^a h_1^b \ \wedge \ y_2 = g_2^a h_2^b\}$$

denotes a proof of knowledge of integers a and b such that $y_1 = g_1^a h_1^b$ and $y_2 = g_2^a h_2^b$ holds, where $y_1, g_1, h_1, y_2, g_2, h_2$ are elements of some groups $G_1 = \langle g_1 \rangle = \langle h_1 \rangle$ and $G_2 = \langle g_2 \rangle = \langle h_2 \rangle$. The variables in the parenthesis denote the values the knowledge of which is being proved, while all other parameters are known to the verifier. Using this notation, a proof of knowledge protocol can be described without getting into all details.

In the random oracle model, such proof of knowledge protocols can be turned into signature schemes using the Fiat-Shamir heuristic [28, 34]. We use the notation $SPK\{(a) : y = z^a\}(m)$ to denote a signature on a message m obtained in this way.

In this paper, we use the following known proof of knowledge protocols:

- Proof of knowledge of discrete logarithms. A proof of knowledge of a discrete logarithm of an element $y \in G$ with respect to a base z is denoted as $PK\{(a) : y = z^a\}$. The discrete logarithms in such proof of knowledge protocol can be modulo a prime [35] or a composite [27, 29], where the composite is a safe-prime product. A proof of knowledge of a representation of an element $y \in G$ with respect to several bases $z_1, \ldots, z_v \in G$ [23] is denoted $PK\{(a_1, \ldots, a_v) : y = z_1^{a_1} \cdot \ldots \cdot z_v^{a_v}\}$.

- Proof of knowledge of equality. A proof of equality of discrete logarithms of two group elements $y_1, y_2 \in G$ to the bases $z_1, z_2 \in G$, respectively, [22, 24] is denoted $PK\{(a) : y_1 = z_1^a \wedge y_2 = z_2^a\}$. Such protocol can also be used to prove that the discrete logarithms of two group elements $y_1 \in G_1$ and $y_2 \in G_2$ to the bases $z_1 \in G_1$ and $z_2 \in G_2$, respectively, in two different groups G_1 and G_2 are equal [9, 16].

- Other various proofs. A proof of knowledge of a discrete logarithm of $y \in G$ with respect to $g \in G$ such that $\log_g y$ lies in the integer interval $[a, b]$ is denoted by $PK\{(x) : y = g^x \ \wedge \ x \in [a, b]\}$. Under the strong RSA assumption, this proof can be done efficiently [6]. Given two existing proof of knowledge protocols, we can efficiently build a proof for the disjunction or conjunction of the knowledge [26].

3.4 Direct Anonymous Attestation

The Direct Anonymous Attestation (DAA) [37, 10] is a scheme that enables remote authentication of a TPM, while preserving the privacy of the user of the platform that contains the TPM. In the DAA protocol, there are an issuer, a platform who has a membership certificate issued by the issuer, and a verifier who wants to get convinced by the platform has a membership certificate. The platform consists of two separate entities: a host and a TPM embedded into the host.

The DAA scheme [10] is constructed from the Camenisch-Lysyanskaya signature scheme [14]: A platform chooses two secret messages f_0 and f_1, obtains a CL signature (membership certificate) on f_0 and f_1 from the issuer via a secure two-party protocol, and then can convince a verifier that it has a membership certificate. This proof to the verifier is performed anonymously by a proof of knowledge of a unique membership certificate. The reason for the platform to choose two messages instead of a single secret message is that it allows the platform to keep the message space small while having enough entropy in the secret, thus achieving better computational performance.

Let us describe the DAA protocol in more details. In the issue protocol, the platform chooses two random ℓ_f-bit secret messages f_0 and f_1, then interacts with the issuer, and in the end obtains (A, e, v) from the protocol such that $A^e R_0^{f_0} R_1^{f_1} S^v \equiv Z \pmod N$. The platform later can prove to a verifier that it has obtained a membership certificate by proving that it got a CL-signature on some values f_0 and f_1. This can be done by a proof of knowledge of values (f_0, f_1, A, e, v) such that $A^e R_0^{f_0} R_1^{f_1} S^v \equiv Z \pmod N$. Let $f = f_0 + f_1 2^{\ell_f}$, the platform also computes $K := B^f \bmod p$ where B is a generator of an algebra group where computing discrete logarithms is infeasible, and proves to the verifier that the exponent f is related to f_0 and f_1 in the proof of knowledge. In the DAA protocol, there are two options to choose B: the value of B can be chosen randomly by the platform, or can be derived from the verifier's name, e.g., using an appropriate hash function.

If a TPM was found comprised and its private key (A, e, f_0, f_1, v) was exposed, the values f_0 and f_1 are extracted and put on a blacklist. The verifier can then check K in the signature against this blacklist by comparing it with $B^{\hat{f}_0 + \hat{f}_1 2^{\ell_f}}$ for all pairs $(\hat{f}_0, \hat{f}_1)$ on the black list. To minimize the computation performed in the TPM in the DAA protocol, some operations for conducting the proof of the unique membership certificate are performed by the host in which the TPM is embedded. This is done without compromising the security of the protocol.

4 Enhanced Privacy ID Scheme

In an Enhanced Privacy ID (EPID) scheme, there are several types of entities: an issuer, a revocation manager, users, and verifiers. The issuer and revocation manager could be the same entity or separate entities. Our EPID scheme builds on top of the DAA scheme [10] and uses the CL signature scheme [14] as underlying building block. To simplify our presentation, we modified the DAA scheme in the following ways: (1) each user chooses a single secret f instead of two secrets, and (2) the signature operation is performed solely by the user, instead of split by two separate entities (e.g., TPM and host in the DAA scheme).

We first briefly describe the basic idea of the EPID scheme before we present the full-fledged EPID protocols. In the join protocol, a user chooses a secret f and sends the issuer a commitment to f, i.e., $U := R^f S^{v'}$, where v' is a value chosen randomly by the user to "blind" the f. Also, the user computes $K := B_I^f \bmod p$, where B_I is a number derived from the issuer's basename. The user sends (K, U) to the issuer and convinces the issuer that K and U are formed correctly. The issuer then issues a membership certificate for the user based on U. The issuer chooses a random integer v'' and a random prime e, then computes A such that $A^e U S^{v''} \equiv Z \pmod N$, and sends the user (A, e, v''). The issuer also proves to the user that he computed A correctly. The CL signature on f is then $(A, e, v := v' + v'')$. The user's private key is set to be (A, e, f, v).

A user can now prove that he is a valid group member by proving that he has a CL signature on some value f. This can be done by a zero-knowledge proof of knowledge of f, A, e, and v such that $A^e R^f S^v \equiv Z \pmod N$. Also, the user computes $K := B^f \bmod p$ where B is a random base picked up by the user, reveals B and K, and proves that $\log_B K$ is the same as the one in his private key. The value K serves the purpose of revocation. Same as in the DAA scheme, if a user's private key (A, e, f, v) is compromised and gets exposed to the public, f is put in the revocation list. The verifier can then check K in the signature against the revocation list by comparing it with $B^{\hat{f}}$ for all $\hat{f}$ in the revocation list. We refer this type of revocation as private-key based revocation and use `priv-RL` to denote the revocation list of this type.

As we mentioned earlier, EPID scheme supports two additional revocation methods, one is signature based revocation and the other is issuer based revocation. In signature based revocation, suppose a verifier received a signature from a prover and then decided that the prover was compromised. The verifier reports the signature to the revocation manager who later places (B, K) of the signature to the signature based revocation list, where $\log_B K$ is the secret of the compromised prover. To prove membership, a user with private key (A, e, f, v) now needs not only to prove the knowledge of (A, e, f, v) such that $A^e R^f S^v \equiv Z \pmod{N}$ but also to prove that f in his private key is different from $\log_{\hat{B}} \hat{K}$ for each $(\hat{B}, \hat{K})$ pair in the signature based revocation list. We use `sig-RL` to denote the revocation list of this type.

Similarly, in issuer based revocation, the issuer obtained (K, U) from a user when the user joined the group and later decided to revoke this user from some reason (e.g., this user left the group). The issuer sends (K, U) to the revocation manager who places K to the issuer based revocation list, where $\log_{B_I} K$ is the secret of the revoked user. To prove membership, a user needs to prove that f in his private key is different from $\log_{B_I} \hat{K}$ for each $\hat{K}$ in the issuer based revocation list. We use `issuer-RL` to denote the revocation list of this type.

4.1 Security Parameters

We now describe the EPID scheme. We use the following security parameters ℓ_N, ℓ_f, ℓ_e, $\ell_{e'}$, ℓ_v, $\ell_\varnothing$, ℓ_H, ℓ_r, ℓ_p, and ℓ_q, where $\ell_N(2048)$ is the size of the RSA modulus, $\ell_f(208)$ is the size of the f's (user's secret, part of membership private key), $\ell_e(576)$ is the size of e's (exponent, part of membership private key), $\ell_{e'}(128)$ is the size of the interval the e's are chosen from, $\ell_v(2720)$ is the size of the v's (random value, part of membership private key), $\ell_\varnothing(80)$ is the security parameter controlling the statistical zero-knowledge property, $\ell_H(256)$ is the output length of the hash function used for Fiat-Shamir heuristic, $\ell_r(80)$ is the security parameter needed for the reduction in the proof of security, $\ell_p(1632)$ is the size of the modulus p, and $\ell_q(208)$ is the size of the order q of the subgroup of $\mathbb{Z}_p^*$ that is used for revocation checking. We require that

$$\ell_\varnothing + \ell_H + 2 + \max\{\ell_f, \ell_{e'}\} < \ell_e, \quad \ell_N + \ell_\varnothing + \ell_H + \max\{\ell_f + \ell_r + 3, \ell_\varnothing + 2\} < \ell_v, \quad \ell_f = \ell_q.$$

The parameters ℓ_p and ℓ_q should chosen such that the discrete logarithm problem in the subgroup of $\mathbb{Z}_p^*$ of order q with p and q being primes such that $p \in [2^{\ell_p - 1}, 2^{\ell_p} - 1]$ and $q \in [2^{\ell_q - 1}, 2^{\ell_q} - 1]$, has about the same difficulty as factoring ℓ_N-bit RSA modulus (e.g., see [31]).

4.2 Setup

This section describes how the issuer chooses the group public key and the group issuing private key. The key generation program also produces a non-interactive proof (using the Fiat-Shamir heuristic [28]) that the public key was formed correctly. The latter will guarantee the security properties of the EPID scheme, i.e., that privacy and anonymity of signatures will hold.

1. The issuer chooses a RSA modulus $N = p_N q_N$ with $p_N = 2p'_N + 1$, $q_N = 2q'_N + 1$ such that p_N, p'_N, q_N, q'_N are all primes, p_N and q_N have the same length, and n has ℓ_N bits.

2. Furthermore, the issuer chooses a random generator g' of QR_N, the group of quadratic residues modulo N.

3. Next, it chooses random integers $x_g, x_h, x_s, x_z, x_r \in [1, p'_N q'_N]$ and computes

$$\begin{array}{lll} g := {g'}^{x_g} \bmod N, & h := {g'}^{x_h} \bmod N, & \\ R := h^{x_r} \bmod N, & S := h^{x_s} \bmod N, & Z := h^{x_z} \bmod N. \end{array}$$

4. It produces a non-interactive proof that g, h, R, S, and Z are computed correctly, i.e., that $g, h \in \langle g' \rangle$ and $S, Z, R \in \langle h \rangle$. This can be proved use the standard cut-and-choose technique. We refer [10] for the details of this proof.

5. The issuer generates a group of prime order as follows: it chooses random primes p and q such that $p = rq+1$ for some r with $q \nmid r$, $p \in [2^{\ell_p - 1}, 2^{\ell_p} - 1]$, and $q \in [2^{\ell_q - 1}, 2^{\ell_q} - 1]$. It then chooses a random $u' \leftarrow \mathbb{Z}_p^*$ such that such that $u'^{(p-1)/q} \not\equiv 1 \pmod p$ and sets $u := u'^{(p-1)/q} \bmod p$.

6. Finally, the issuer publishes the group public key $(N, g', g, h, R, S, Z, p, q, u)$ and the proof, and stores (p'_N, q'_N) as the group issuing private key.

In addition to generating the group public key and group issuing private key, the issuer generates also a long-term public/private key pair (K_I, K_I^{-1}). The issuer publishes the public key K_I. This key is used for authentication between the issuer and any user who wants to become a group member. Analogously, the revocation manager has a long-term public/private key pair (K_R, K_R^{-1}). The revocation manager uses its key to sign the revocation list.

4.3 Verification of the Issuer's Public Key

Given the group public key $(N, g', g, h, R, S, Z, p, q, u)$ and the proof that g, h, S, Z, R are formed properly, any user in the system can verify the correctness of the group public key as follows:

1. Verify the proof that $g, h \in \langle g' \rangle$ and $R, S, Z \in \langle h \rangle$.

2. Check whether p and q are primes, $q \mid (p-1)$, $q \nmid \frac{p-1}{q}$, and $u^q \equiv 1 \pmod p$.

3. Check whether all public key parameters have the required length.

If g, h, R, S, Z are not formed correctly, it could potentially mean that the security properties for the users do not hold. However, it is sufficient if the users verify the proof that g, h, R, S, Z are computed correctly only once. Also, if u does not generate a subgroup of $\mathbb{Z}_p^*$, the issuer could potentially use this to link different signatures. As argued in [10], it is not necessary to prove that N is a product of two safe primes for the anonymity of the users. In fact, it would be very expensive for the issuer to prove that N is a safe-prime product [15].

4.4 Join Protocol

The join protocol is a protocol runs between the issuer and a user. The public input to this protocol is the group public key $(N, g', g, h, R, S, Z, p, q, u)$ and the issuer's long-term public key K_I and the issuer's basename $\mathtt{bsn}_I$. The private input of the issuer is the group issuing private key (p_N, q_N). We assume that the user and the issuer have established an authentic channel, i.e., the user needs to make sure that he talks to the right issuer and the issuer needs to be sure that the user is allowed to join the group. Note that we do not require secrecy of the communication channel.

Let $H(\cdot)$ and $H_p(\cdot)$ be two collision-resistant hash functions $H(\cdot) : \{0,1\}^* \rightarrow \{0,1\}^{\ell_H}$ and $H_p(\cdot) : \{0,1\}^* \rightarrow \{0,1\}^{\ell_p + \ell_\varnothing}$. The join protocol takes the following steps:

1. The user verifies that the group public key $(N, g', g, h, R, S, Z, p, q, u)$ is authenticated by K_I.
2. Both the user and issuer compute $B_I := H_p(\mathtt{bsn}_I)^{(p-1)/q} \bmod p$.
3. The user chooses at random

$$f \leftarrow \mathbb{Z}_q^*, \qquad v' \leftarrow \{0,1\}^{\ell_N+\ell_\varnothing},$$

then computes

$$K := B_I^f \bmod p, \qquad U := R^f S^{v'} \bmod N.$$

The user sends (K, U) to the issuer.

4. The user proves to the issuer the knowledge of f and v'. He runs as the prover the protocol

$$SPK\{(f, v') : \ U \equiv R^f S^{v'} \pmod N \ \wedge \ K \equiv B_I^f \pmod p \ \wedge \ f \in \{0,1\}^{\ell_f+\ell_\varnothing+\ell_H+1} \ \wedge \ v' \in \{0,1\}^{\ell_N+\ell_\varnothing+\ell_H+1}\}(n_I)$$

with the issuer as the verifier. We use Σ to denote the "signature of knowledge" of the above protocol. The issuer records (K, U, Σ) in its database, so that he has the ability to revoke this user in the future, see Section 4.6 for detailed discussion. This protocol is implemented as follows.

   (a) The issuer chooses a random string $n_I \leftarrow \{0,1\}^{\ell_H}$ and sends n_I to the user.

   (b) The user chooses at random

$$r_f \leftarrow \{0,1\}^{\ell_f+\ell_\varnothing+\ell_H}, \qquad r_{v'} \leftarrow \{0,1\}^{\ell_N+2\ell_\varnothing+\ell_H},$$

   and computes

$$\tilde{K} := B_I^{r_f} \bmod p, \qquad \tilde{U} := R^{r_f} S^{r_{v'}} \bmod N.$$

   (c) The user computes $c := H(N\|R\|S\|B_I\|K\|U\|\tilde{K}\|\tilde{U}\|n_I)$.

   (d) The user computes

$$s_f := r_f + c \cdot f, \qquad s_{v'} := r_{v'} + c \cdot v'.$$

   and sends $(c, s_f, s_{v'})$ to the issuer. The "signature of knowledge" is $\Sigma = (c, s_f, s_{v'})$.

   (e) The issuer verifies the proof as follows. The issuer computes

$$\hat{K} := K^{-c} B_I^{s_f} \bmod p, \qquad \hat{U} := U^{-c} R^{s_f} S^{s_{v'}} \bmod N,$$

   and checks that

$$s_f \overset{?}{\in} \{0,1\}^{\ell_f+\ell_\varnothing+\ell_H+1}, \qquad s_{v'} \overset{?}{\in} \{0,1\}^{\ell_N+2\ell_\varnothing+\ell_H+1},$$

   and

$$c \overset{?}{=} H(N\|R\|S\|B_I\|K\|U\|\hat{K}\|\hat{U}\|n_I).$$

5. The issuer chooses a random $v'' \leftarrow [2^{\ell_v - 1}, 2^{\ell_v} - 1]$ and a random prime $e \leftarrow [2^{\ell_e}, 2^{\ell_e} + 2^{\ell_{e'}}]$ and computes
$$A := \left(\frac{Z}{US^{v''}}\right)^{1/e} \bmod N.$$

6. To convince the user that A was correctly computed, the issuer as prover runs the protocol
$$SPK\{(d) \ : \ A \equiv \left(\frac{Z}{US^{v''}}\right)^{d} \pmod N\}(n_U)$$
with the host:
   (a) The user chooses a random integer $n_U \leftarrow \{0,1\}^{\ell_H}$ and sends n_U to the issuer.
   (b) The issuer randomly chooses $r_e \leftarrow [0, p'_N q'_N]$ and computes
$$\tilde{A} := \left(\frac{Z}{US^{v''}}\right)^{r_e} \bmod N,$$
   and
$$c' := H(N\|Z\|S\|U\|v''\|A\|\tilde{A}\|n_U), \qquad s_e := r_e + c'/e \bmod p'_N q'_N,$$
   and sends c', s_e, and (A, e, v'') to the user.
   (c) The user verifies whether e is a prime and lies in $[2^{\ell_e}, 2^{\ell_e} + 2^{\ell_{e'}}]$, computes
$$\hat{A} := A^{-c'} \left(\frac{Z}{US^{v''}}\right)^{s_e} \bmod N,$$
   and checks whether $c' \overset{?}{=} H(N\|Z\|S\|U\|v''\|A\|\hat{A}\|n_U)$.

7. The user sets $v := v'' + v'$ and stores (A, e, f, v) as its membership private key.

Same as in the DAA scheme [10], the issuer proves to the user that A was formed correctly, i.e., A lies in $\langle h \rangle$. In Step 6 of the above protocol, the issuer proves that $A \equiv (ZU^{-1}S^{-v''})^d \pmod N$ for some value d. In the setup program, the issuer proves that $S, Z, R \in \langle h \rangle$. Since $U = R^f S^{v'} \bmod N$, the user can conclude that $A \in \langle h \rangle$. The reason for requiring $A \in \langle h \rangle$ is to assure that later, in the proof of membership protocol, A can be statistically hidden in $\langle h \rangle$. Otherwise, an adversarial issuer could link signatures generated by users whose A does not lie in $\langle h \rangle$. Note that schemes such as [1, 12, 14] have prevented this by ensuring that N is a safe-prime product and then made sure that all elements are members of QR_N. However, proving that a modulus is a safe-prime product is rather inefficient [15] and hence the setup of these schemes is not practical as our scheme.

4.5 Proof of Membership Protocol

The proof of membership protocol is a protocol run by a prover and a verifier. It contains the following four steps: request, challenge, sign, and verify. In the request step, the prover initializes the interaction with the verifier by sending a request to the verifier. We shall describe rest of the three steps in details.

As we mentioned earlier, there are three types of revocation: private-key based revocation, signature based revocation, and issuer based revocation. Therefore, the revocation list `RL` contains three sublists, i.e., $\mathtt{RL} = \{\mathtt{priv\text{-}RL}, \mathtt{sig\text{-}RL}, \mathtt{issuer\text{-}RL}\}$. Let `priv-RL` be the revocation list for private-key based revocation, in which each element is a value in $\langle u \rangle$. Let `sig-RL` be the revocation list for signature based revocation, in which each element is a pair of values in $\langle u \rangle$. Let `issuer-RL` be the revocation list for issuer based revocation, in which each element is a value in $\langle u \rangle$. The revocation manager maintains the revocation list and regularly publishes the newest revocation list to everyone in the system, signed using his private key. That is, the revocation manager publishes $\{\mathtt{priv\text{-}RL}\}_{K_R^{-1}}$, $\{\mathtt{sig\text{-}RL}\}_{K_R^{-1}}$, and $\{\mathtt{issuer\text{-}RL}\}_{K_R^{-1}}$.

4.5.1 Challenge

In this step, the verifier first chooses a message m and a nonce $n_V \leftarrow \{0,1\}^{\ell_H}$. The verifier then sends to the prover m, n_V, $\{\mathtt{sig\text{-}RL}\}_{K_R^{-1}}$, and $\{\mathtt{issuer\text{-}RL}\}_{K_R^{-1}}$ as the challenge. After the prover receives the challenges from the verifier, the prover verifies the content of `sig-RL` and `issuer-RL` using the revocation manager's public key K_R. Let (A, e, f, v) be the prover's private key. For each element (B_i, K_i) in `sig-RL`, the prover checks whether $B_i^f \not\equiv K_i \pmod p$. If there exists some i such that $B_i^f \equiv K_i \pmod p$, it means that the prover has been revoked, the prover aborts the proof of membership protocol. Analogously, for each item K_i in `issuer-RL`, the prover checks whether $B_I^f \not\equiv K_i \pmod p$ where B_I is the base derived from the issuer's basename $\mathtt{bsn}_I$. The prover quits the proof of membership protocol if the check fails.

Note that the prover can directly obtain `RL` from the revocation manager and checks whether he has been revoked. However, it is not required for the prover to conduct such operation. Also note that it is the verifier's responsibility to obtain the latest revocation list from the revocation manager. If `sig-RL` and `issuer-RL` in the verifier's challenge are not the latest ones, then there is a chance that some revoked users may successfully perform membership proof to the verifier without being detected.

4.5.2 Sign

This step is run by the prover. The input to this program is the group public key $(N, g', g, h, R, S, Z, p, q, u)$, the prover's private key (A, e, f, v), the verifier's message m and nonce n_V, the signature based revocation list `sig-RL`, and the revocation based revocation list `issuer-RL`. The output to this program is a signature σ produced by the prover. The sign program takes the following steps.

1. The prover picks a random $B \leftarrow \langle u \rangle$ and two integers $w, r \leftarrow \{0,1\}^{\ell_N + \ell_\varnothing}$ and computes

$$T_1 := Ah^w \bmod N, \qquad T_2 := g^w h^e (g')^r \bmod N, \qquad K := B^f \bmod p.$$

2. The prover produces a "signature of knowledge" that T_1 and T_2 are commitments to the prover's private key and K was computed using the prover's secret f. That is, the prover computes the "signature of knowledge"

$$\begin{aligned} SPK\{(f, v, e, w, r, ew, ee, er) : Z \equiv T_1^e R^f S^v h^{-ew} \pmod N \wedge \\ T_2 \equiv g^w h^e (g')^r \pmod N \ \wedge\ 1 \equiv T_2^{-e} g^{ew} h^{ee} (g')^{er} \pmod N \wedge \\ K \equiv B^f \pmod p \ \wedge\ f \in \{0,1\}^{\ell_f + \ell_\varnothing + \ell_H + 1} \ \wedge\ (e - 2^{\ell_e}) \in \{0,1\}^{\ell_{e'} + \ell_\varnothing + \ell_H + 1}\}(n_V \| m) \end{aligned}$$

with the following steps.

(a) The prover picks random integers

$$\begin{aligned} r_v &\leftarrow \{0,1\}^{\ell_v+\ell_\varnothing+\ell_H}, & r_f &\leftarrow \{0,1\}^{\ell_f+\ell_\varnothing+\ell_H} \\ r_e &\leftarrow \{0,1\}^{\ell_{e'}+\ell_\varnothing+\ell_H}, & r_{ee} &\leftarrow \{0,1\}^{\ell_e+\ell_\varnothing+\ell_H+1}, \\ r_w, r_r &\leftarrow \{0,1\}^{\ell_N+2\ell_\varnothing+\ell_H}, & r_{ew}, r_{er} &\leftarrow \{0,1\}2^{\ell_e+\ell_N+2\ell_\varnothing+\ell_H+1}. \end{aligned}$$

(b) The prover computes

$$\begin{aligned} \tilde{T}_1 &:= T_1^{r_e} R^{r_f} S^{r_v} h^{-r_{ew}} \bmod N, & \tilde{T}_2 &:= g^{r_w} h^{r_e} (g')^{r_r} \bmod N, \\ \tilde{T}_3 &:= T_2^{-r_e} g^{r_{ew}} h^{r_{ee}} (g')^{r_{er}} \bmod N, & \tilde{K} &:= B^{r_f} \bmod p. \end{aligned}$$

(c) The prover computes

$$c_1 := H(N\|g'\|g\|h\|R\|S\|Z\|p\|q\|u\|B\|K\|T_1\|T_2\|\tilde{T}_1\|\tilde{T}_2\|\tilde{T}_3\|\tilde{K}\|m\|n_V).$$

(d) The prover computes (over the integers)

$$\begin{aligned} s_v &:= r_v + c_1 \cdot v, & s_f &:= r_f + c_1 \cdot f, & & \\ s_e &:= r_e + c_1 \cdot (e - 2^{\ell_e}), & s_r &:= r_r + c_1 \cdot r, & s_w &:= r_w + c_1 \cdot w, \\ s_{ew} &:= r_{ew} + c_1 \cdot w \cdot e, & s_{ee} &:= r_{ee} + c_1 \cdot e^2, & s_{er} &:= r_{er} + c_1 \cdot e \cdot r. \end{aligned}$$

(e) The prover sets $\sigma_1 := (B, K, T_1, T_2, c_1, s_v, s_f, s_e, s_r, s_w, s_{ew}, s_{ee}, s_{er})$.

3. The prover produces a "signature of knowledge" that his private key has not been revoked in `sig-RL`. Let `sig-RL` $= \{(B_1, K_1), \ldots, (B_{n_2}, K_{n_2})\}$. The prover computes the "signature of knowledge"

$$SPK\{(f) \ : \ K \equiv B^f \pmod p \ \wedge \ K_1 \not\equiv B_1^f \pmod p \ \wedge \ \ldots \ \wedge \ K_{n_2} \not\equiv B_{n_2}^f \pmod p\}(n_V\|m)$$

with the following steps.

(a) The prover chooses a random $r \leftarrow \mathbb{Z}_q$ and computes $\tilde{K} := B^r \bmod p$.

(b) For $i = 1, \ldots, n_2$, the prover does the following:

i. The prover chooses a random $x_i \leftarrow \mathbb{Z}_q$.

ii. The prover computes

$$U_i := B_i^{x_i} \bmod p, \qquad V_i := K_i^{x_i} \bmod p, \qquad W_i := U_i^f \bmod p.$$

iii. The prover chooses a random integer $r_i \leftarrow \mathbb{Z}_q$.

iv. The prover computes

$$\tilde{U}_i := B_i^{r_i} \bmod p, \qquad \tilde{V}_i := K_i^{r_i} \bmod p, \qquad \tilde{W}_i := U_i^r \bmod p.$$

(c) The prover computes

$$\begin{aligned} c_2 := H(p\|q\|u\|B\|K\|\tilde{K}\|U_1\|V_1\|W_1\|\tilde{U}_1\|\tilde{V}_1\|\tilde{W}_1\| \ldots \\ \|U_{n_2}\|V_{n_2}\|W_{n_2}\|\tilde{U}_{n_2}\|\tilde{V}_{n_2}\|\tilde{W}_{n_2}\|m\|\texttt{sig-RL}\|n_V). \end{aligned}$$

(d) For $i = 1, \ldots, n_2$, the prover computes $s_i := r_i + c_2 \cdot x_i \bmod q$.

(e) The prover computes $s := r + c_2 \cdot f \bmod q$.

(f) The prover sets $\sigma_2 := (B, K, c_2, s, U_1, V_1, W_1, s_1, \ldots, U_{n_2}, V_{n_2}, W_{n_2}, s_{n_2})$.

4. The prover produces a "signature of knowledge" that his private key has not been revoked in `issuer-RL`. Let `issuer-RL` $= \{B_I, K_1, \ldots, K_{n_3}\}$. The prover computes the "signature of knowledge"

$$SPK\{(f) \ : \ K \equiv B^f \pmod p \ \wedge \ K_1 \not\equiv B_I^f \pmod p \ \wedge \ \ldots \ \wedge \ K_{n_3} \not\equiv B_I^f \pmod p\}(n_V \| m)$$

with the following steps.

(a) The prover chooses a random $x \leftarrow \mathbb{Z}_q$ and computes

$$U := B_I^x \bmod p, \qquad W := U^f \bmod p.$$

(b) For $i = 1, \ldots, n_3$, the prover computes $V_i := K_i^x \bmod p$.

(c) The prover chooses $r_f \leftarrow \mathbb{Z}_q$ and $r_x \leftarrow \mathbb{Z}_q$ and computes

$$\tilde{U} := U^{r_x} \bmod p, \qquad \tilde{W} := W^{r_f} \bmod p, \qquad \tilde{K} := B^{r_f} \bmod p.$$

(d) For $i = 1, \ldots, n_3$, the prover computes $\tilde{V}_i := K_i^{r_x} \bmod p$.

(e) The prover computes

$$c_3 := H(p\|q\|u\|B\|K\|\tilde{K}\|U\|\tilde{U}\|V_1\|\tilde{V}_1\| \ldots \|V_{n_3}\|\tilde{V}_{n_3}\|W\|\tilde{W}\|m\|\texttt{issuer-RL}\|n_V).$$

(f) The prover computes

$$s_x := r_x + c_3 \cdot x \bmod q, \qquad s_f := r_f + c_3 \cdot f \bmod q.$$

(g) The prover sets $\sigma_3 := (B, K, c_3, s_x, s_f, U, V_1, \ldots, V_{n_3}, W)$.

5. The prover outputs the signature $\sigma := (\sigma_1, \sigma_2, \sigma_3)$ and sends σ to the verifier.

Observe that in the sign process, the prover proves the knowledge of f such that $B^f \equiv K \pmod p$ three times, one in each "signature of knowledge". We could merge all three "signatures of knowledge" together such that the prover only needs to prove the knowledge of f once, thus could improve the performance of proof of membership slightly. When we present the above sign process, we choose to have three separate proof of knowledge protocols to make our protocol easier to read.

4.5.3 Verify

Given the group public key $(N, g', g, h, R, S, Z, u, p, q)$, the message m, the nonce n_V, the corresponding signature $\sigma = (\sigma_1, \sigma_2, \sigma_3)$, and the revocation list `RL` $= \{$`priv-RL`, `sig-RL`, `issuer-RL`$\}$, the verifier verifies the signature as follows.

1. The verifier verifies that m and n_V are the message and the nonce he sent to the prover in the challenge step. The verifier also verifies (B, K) in σ_1, σ_2, and σ_3 all matches.

2. The verifier verifies the correctness of $\sigma_1 = (B, K, T_1, T_2, c_1, s_v, s_f, s_e, s_r, s_w, s_{ew}, s_{ee}, s_{er})$ as follows:

(a) The verifier computes $s'_e := s_e + c_1 \cdot 2^{\ell_e}$ and computes

$$\hat{T}_1 := Z^{-c_1} T_1^{s'_e} R^{s_f} S^{s_v} h^{-s_{ew}} \bmod N, \qquad \hat{T}_2 := T_2^{-c_1} g^{s_w} h^{s'_e} (g')^{s_r} \bmod N,$$
$$\hat{T}_3 := T_2^{-s'_e} g^{s_{ew}} h^{s_{ee}} (g')^{s_{er}} \bmod N, \qquad \hat{K} := K^{-c_1} B^{s_f} \bmod p.$$

(b) The verifier verifies that

$$B, K \overset{?}{\in} \langle u \rangle, \qquad s_f \overset{?}{\in} \{0,1\}^{\ell_f + \ell_\varnothing + \ell_H + 1}, \qquad s_e \overset{?}{\in} \{0,1\}^{\ell_{e'} + \ell_\varnothing + \ell_H + 1}.$$

(c) The verifier verifies that

$$c_1 \overset{?}{=} H(N \| g' \| g \| h \| R \| S \| Z \| p \| q \| u \| B \| K \| T_1 \| T_2 \| \hat{T}_1 \| \hat{T}_2 \| \hat{T}_3 \| \hat{K} \| m \| n_V).$$

3. The verifier verifies that the prover's private key has not been revoked in `priv-RL`, where `priv-RL` $= \{f_1, \ldots, f_{n_1}\}$. For $i = 1, \ldots, n_1$, the verifier verifies that

$$K \overset{?}{\not\equiv} B^{f_i} \pmod{p}.$$

4. The verifier verifies the correctness of $\sigma_2 = (B, K, c_2, s, U_1, V_1, W_1, s_1, \ldots, U_{n_2}, V_{n_2}, W_{n_2}, s_{n_2})$ based on `sig-RL` $= \{(B_1, K_1), \ldots, (B_{n_2}, K_{n_2})\}$. It takes the following steps:

(a) The verifier computes $\hat{K} := K^{-c_2} B^s \bmod p$.

(b) For $i = 1, \ldots, n_2$, the verifier does the following:

i. The verifier verifies that

$$U_i, V_i, W_i \overset{?}{\in} \langle u \rangle, \qquad s_i \overset{?}{\in} \mathbb{Z}_q, \qquad V_i \overset{?}{\neq} W_i.$$

ii. The verifier computes

$$\hat{U}_i := U_i^{-c_2} B_i^{s_i} \bmod p, \qquad \hat{V}_i := V_i^{-c_2} K_i^{s_i} \bmod p, \qquad \hat{W}_i := W_i^{-c_2} U_i^{s} \bmod p.$$

(c) The verifier verifies that

$$c_2 \overset{?}{=} H(p \| q \| u \| B \| K \| \hat{K} \| U_1 \| V_1 \| W_1 \| \hat{U}_1 \| \hat{V}_1 \| \hat{W}_1 \| \ldots$$
$$\| U_{n_2} \| V_{n_2} \| W_{n_2} \| \hat{U}_{n_2} \| \hat{V}_{n_2} \| \hat{W}_{n_2} \| m \| \texttt{sig-RL} \| n_V).$$

5. The verifier verifies the correctness of $\sigma_3 = (B, K, c_3, s_x, s_f, U, V_1, \ldots, V_{n_3}, W)$ based on `issuer-RL` $= \{B_I, K_1, \ldots, K_{n_3}\}$. It takes the following steps:

(a) The verifier verifies that

$$U, W \overset{?}{\in} \langle u \rangle, \qquad s_x, s_f \overset{?}{\in} \mathbb{Z}_q.$$

(b) The verifier computes

$$\hat{K} := K^{-c_3} B^{s_f} \bmod p, \qquad \hat{U} := U^{-c_3} B_I^{s_x} \bmod p, \qquad \hat{W} := W^{-c_3} U^{s_f} \bmod p.$$

(c) For $i = 1, \ldots, n_3$, the verifier does the following:

i. The verifier verifies that

$$V_i \overset{?}{\in} \langle u \rangle, \qquad\qquad V_i \overset{?}{\neq} W.$$

ii. The verifier computes $\hat{V}_i := V_i^{-c_3} K_i^{s_x} \bmod p$.

(d) The verifier verifies that

$$c_3 \overset{?}{=} H(p\|q\|u\|B\|K\|\hat{K}\|U\|\hat{U}\|V_1\|\hat{V}_1\|\ldots\|V_{n_3}\|\hat{V}_{n_3}\|W\|\hat{W}\|m\|\texttt{issuer-RL}\|n_V).$$

6. If all the above verifications succeed, the verifier outputs `succeed`, otherwise outputs `fail`.

Note that the verifier can apply so called batch verification techniques [3] to obtain a considerable speed-up of the verification in step 3.

4.6 Revocation

There are three sublists in the revocation list: `priv-RL`, `sig-RL`, and `issuer-RL`. Initially, `priv-RL` and `sig-RL` are set to be empty, and `issuer-RL` is set to be $\{B_I\}$, where $B_I \equiv H_p(\texttt{bsn}_I)^{(p-1)/q} \bmod p$ and $\texttt{bsn}_I$ is the issuer's basename. There are three ways to revoke a group member. We describe each of these ways in detail.

1. When a user is compromised and his private key (A, e, f, v) has been exposed (e.g., on the Internet or embedded into some software), the revocation manager verifies the correctness of this exposed key by checking $A^e R^f S^v \equiv Z \pmod{N}$, then adds f to `priv-RL`.

2. When a verifier interacts with some compromised prover and finds the prover suspicious, the verifier reports the prover's signature $\sigma = (\sigma_1, \sigma_2, \sigma_3)$ along with some other physical evidences to the revocation manager. After the revocation manager verifies the physical evidences and correctness of σ_1 (see Step 2 of Section 4.5.3), he adds (B, K) in σ_1 to `sig-RL`.

3. When the issuer wants to revoke a group member (e.g., because that user leaves the group), the issuer sends (K, U, Σ) to the revocation manager, where the (K, U, Σ) tuple was obtained from the "to-be-revoked" user during step 4 of the join protocol. The revocation manager verifies that correctness of Σ and then adds K to `issuer-RL`.

Note that when the revocation manager revokes a user based on the signature of the user, he needs to make sure that the signature is valid, i.e., the signature was signed by a group member. This is to prevent a malicious verifier from adding arbitrary (B, K) pair to `sig-RL`. Similarly, when the revocation manager revokes a user based on (K, U, Σ) from the issuer, he needs to make sure that Σ is a correct "signature of knowledge". This is to prevent the (malicious) issuer from adding arbitrary K to `issuer-RL`. Observe that, the issuer can always add new members, create new signatures, and later revoke the members that he created by herself. However, even though the malicious issuer can choose K of his choice, he has to know $\log_B K$ in order to create a valid signature σ or know $\log_{B_I} K$ to create a valid Σ. This is a requirement in our security proof (see Section 6 for details).

After the revocation manager publishes the revocation list `RL` and signs using his private key K_R^{-1}, everyone can verify the authenticity of this revocation list using the revocation manager's public key K_R. In practice, we may assume that the revocation manager is trusted. Then the verifiers trust the revocation manager to construct the revocation list in a correct manner. In the model where the revocation manager is not completely trusted, the revocation manager also needs to publish a compromised private key for each item in `priv-RL`, a signature for each item in `sig-RL`, and a (K, U, Σ) tuple for each element in `issuer-RL`. The verifiers have to verify the correctness of each element in the revocation list in the same way as the revocation manager does. In Section 6, we shall show that that even if the revocation manager or the issuer has been corrupted by the adversary, the anonymity of the honest users is still guaranteed.

4.7 Performance and Discussion

The setup and join protocol have the same performance as in the DAA scheme [10]. The cost of proof of membership protocol has four parts: proof of knowledge of a membership private key, verification that the private key is not in `priv-RL`, proof that the private key does not appear in `sig-RL`, and proof that the private key does not appear in `issuer-RL`. The first part of the proof of membership protocol is the same as the DAA scheme and takes constant time for both the prover and verifier. The second part is also the same as the DAA scheme and takes n_1 modular exponentiations for the verifier, where n_1 is the size of `priv-RL`. The third and forth parts together take about $6n_2 + 2n_3 + c$ modular exponentiations for both the prover and verifier, where n_2 and n_3 are the lengths of `sig-RL` and `issuer-RL`, respectively, and c is a small constant.

Observe that the cost of proof of membership is linear to the size of the revocation list and could be quite expensive if the revocation list becomes large. There are two possible ways to control the size of the revocation list.

- Divide into smaller groups. If the group size is too big, the revocation list may become large as well. One way is to control the size of the revocation list is to have multiple smaller groups. If a group size was 10,000, and at most 2% of the users would get revoked, then the revocation list would have at most 200 items. The drawback of this method is that the verifier needs to know which group the prover is in, thus, learns more information about the prover. It is a trade-off between privacy and performance.

- Issue a new group if the revocation list grows too big. If the size of the revocation list is above certain threshold (e.g., 2% of the group size), then the issuer can do a "re-key" process as follows. The issuer first creates a new group. Then each user in the old group proves to the issuer that he is a legitimate member of the old group and has not been revoked, then obtains a new membership private key for the new group.

5 Intuition

In the EPID scheme in the previous section, the prover needs to perform non-revoked proofs, i.e., prove that his private key is not listed in `sig-RL` and `issuer-RL`. The non-revoked proofs are derived, via Fiat-Shamir heuristic [28], from a new protocol for proving inequality of discrete logarithms. We first present the inequality proof protocol, then give intuition into the construction of the non-revoked proofs in the EPID scheme.

Loosely speaking, given $B, K, B_i, K_i \in \langle u \rangle$, the prover proves the knowledge of f such that $B^f = K$ and $B_i^f \neq K_i$. The protocol below is a proof of knowledge, which means that by rewinding a prover it is possible to extract the secret f. The protocol itself however is not zero knowledge.

Protocol 1. Let p and q be two large primes such that $p = rq + 1$ for some r with $q \nmid r$. Let u be a generator of the unique order-q subgroup of $\mathbb{Z}_p^*$. Assume all the arithmetic operations in this section are modulo p unless specified otherwise. The prover and the verifier have common input

$B, K, B_i, K_i \in \langle u \rangle$. The prover has the additional input f such that $B^f = K$. The prover wants to prove the knowledge of f such that $f = \log_B K$ and $f \neq \log_{B_i} K_i$, i.e.,

$$PK\{(f) \ : \ B^f = K \ \wedge \ B_i^f \neq K_i\}$$

The prover and the verifier engage in the following protocol.

1. The prover chooses $x \leftarrow \mathbb{Z}_q$ and computes

$$U := B_i^x, \qquad V := K_i^x; \qquad W := U^f.$$

   The prover sends U, V, and W to the verifier.

2. The prover proves to the verifier

$$PK\{(x, f) \ : \ B_i^x = U \ \wedge \ K_i^x = V \ \wedge \ U^f = W \ \wedge \ B^f = K\}$$

   as follows:

   (a) The prover chooses $r_x \leftarrow \mathbb{Z}_q$ and $r_f \leftarrow \mathbb{Z}_q$, and computes

$$\tilde{U} := B_i^{r_x}, \qquad \tilde{V} := K_i^{r_x}; \qquad \tilde{W} := U^{r_f}, \qquad \tilde{K} := B^{r_f}.$$

   (b) The prover sends $\tilde{U}$, $\tilde{V}$, $\tilde{W}$, and $\tilde{K}$ to the verifier.

   (c) The verifier chooses a random challenge $c \leftarrow \mathbb{Z}_q$ and sends c back to the prover.

   (d) The prover computes

$$s_x := r_x + c \cdot x \bmod q, \qquad s_f := r_f + c \cdot f \bmod q$$

   The prover then sends s_x and s_f to the verifier.

   (e) The verifier verifies that

$$B_i^{s_x} \stackrel{?}{=} \tilde{U} \cdot U^c, \qquad K_i^{s_x} \stackrel{?}{=} \tilde{V} \cdot V^c, \qquad U^{s_f} \stackrel{?}{=} \tilde{W} \cdot W^c, \qquad B^{s_f} \stackrel{?}{=} \tilde{K} \cdot K^c. \quad (1)$$

3. The verifier verifies that $V \neq W$.

Let us use f_i to denote $\log_{B_i} K_i$. Suppose $f = f_i$ and the proof of knowledge in Step 2 of the above protocol is correct, then we have $V = K_i^x = B_i^{xf_i} = B_i^{xf}$ and $W = U^f = B_i^{xf}$ for some x and f, therefore $V = W$ and the verifier will reject the above inequality proof. Protocol 1 is efficient: the prover needs to perform 7 modular exponentiations and the verifier needs to perform 8 modular exponentiations. Note that Camenisch and Shoup [17] have developed a zero-knowledge proof of knowledge protocol for proving inequality of discrete logarithms. Their protocol has similar computational complexity as our protocol, and furthermore their protocol is zero-knowledge whereas our protocol is not zero-knowledge. Our EPID scheme uses derived versions of Protocol 1 because our protocol has a significant advantage in proving multiple inequality equations at a time.

Lemma 1. *Given (U, V, W), transcripts of Protocol 1 can be simulated.*

Proof. The simulator chooses the challenge $c \leftarrow \mathbb{Z}_q$. It selects $s_x \leftarrow \mathbb{Z}_q$ and sets $\tilde{U} := B_i^{s_x} U^{-c}$ and $\tilde{V} := K_i^{s_x} V^{-c}$. Then the first two equations in (1) are satisfied. With c and x fixed, a choice of either r_x or s_x determines the other, and a uniform random choice of one gives a uniform random choice of the other. Therefore, s_x, $\tilde{U}$, and $\tilde{V}$ are distributed as in a real transcript.

The simulator now chooses $s_f \leftarrow \mathbb{Z}_q$ and sets $\tilde{W} := U^{s_f} W^{-c}$ and $\tilde{K} := B^{s_f} K^{-c}$. Then the last two equations in (1) are satisfied. Since c and f are fixed, a choice of either r_f or s_f determines the other. Therefore, s_f, $\tilde{W}$, and $\tilde{K}$ are distributed as in a real transcript.

Finally, the simulator outputs the transcript $(U, V, W, \tilde{U}, \tilde{V}, \tilde{W}, \tilde{K}, c, s_x, s_f)$. As argue above, this transcript is distributed identically to the transcript of Protocol 1 for given (U, V, W). □

Lemma 2. *There exists a knowledge extractor for Protocol 1 that can extract an f from a convincing prover, such that $B^f = K$ and $B_i^f \neq K_i$.*

Proof. Suppose that a knowledge extractor can rewind a prover in the protocol above. The prover sends U, V, W, $\tilde{U}$, $\tilde{V}$, $\tilde{W}$, and $\tilde{K}$ to the verifier, where $V \neq W$. To challenge value c, the prover responds with s_x and s_f. To challenge value $c' \neq c$, the prover responds with s'_x and s'_f. If the prover is convincing, all four verification equations in (1) holds for both (s_x, s_f) and (s'_x, s'_f).

For simplicity, we denote $\Delta c = c - c'$, $\Delta s_x = s_x - s'_x$, and $\Delta s_f = s_f - s'_f$. Consider equations (1) in Protocol 1, dividing each equation using (c, s_x, s_f) and using (c', s'_x, s'_f), we obtain

$$B_i^{\Delta s_x} = U^{\Delta c}, \qquad K_i^{\Delta s_x} = V^{\Delta c}, \qquad U^{\Delta s_f} = W^{\Delta c}, \qquad B^{\Delta s_f} = K^{\Delta c}.$$

The exponents are in a group of prime order q, therefore we can take roots. Let

$$\hat{x} := \Delta s_x / \Delta c \bmod q, \qquad \hat{f} := \Delta s_f / \Delta c \bmod q.$$

We have the following equations:

$$B_i^{\hat{x}} = U, \qquad K_i^{\hat{x}} = V, \qquad U^{\hat{f}} = W, \qquad B^{\hat{f}} = K. \tag{2}$$

If we combine the first and third equations in (2), we get

$$B_i^{\hat{x}\cdot\hat{f}} = W, \qquad K_i^{\hat{x}} = V.$$

After we take $\hat{x}$-th root to both sides of the above equations, we obtain

$$B_i^{\hat{f}} = W^{1/\hat{x}}, \qquad K_i = V^{1/\hat{x}}.$$

As $V \neq W$, it follows that $V^{1/\hat{x}} \neq W^{1/\hat{x}}$. Therefore we have $B_i^{\hat{f}} \neq K_i$. That is, the knowledge extractor obtains $\hat{f}$ such that $B^{\hat{f}} = K$ and $B_i^{\hat{f}} \neq K_i$. Given $B, K \in \langle u \rangle$, there exists only one $f \in \mathbb{Z}_q$ such that $B^f = K$. Thus the $\hat{f}$ extracted by the knowledge extractor is the same as the f known to the prover. □

Note that in the EPID scheme, given `sig-RL` and `issuer-RL`, the prover wants to prove the knowledge of f such that $B^f = K$, $B_i^f \neq K_i$ for each (B_i, K_i) pair in `sig-RL`, and $B_I^f \neq K_i$ for each K_i in `issuer-RL`. To do this, one could repeat Protocol 1 for multiple times, once for each item in `sig-RL` and `issuer-RL`. In the EPID scheme, we use Protocol 2 for proving that f is not revoked in `sig-RL` and Protocol 3 for proving that f is not revoked in `issuer-RL`, the following two protocols are derived from Protocol 1.

Protocol 2. Let p, q, and u be defined as in Protocol 1. The prover and the verifier have common input $B, K, B_1, K_1, \ldots, B_n, K_n \in \langle u \rangle$. The prover has the additional input f such that $B^f = K$. The prover wants to prove

$$PK\{(f) \; : \; B^f = K \; \wedge \; B_1^f \neq K_1 \; \wedge \; \cdots \; \wedge \; B_n^f \neq K_n\}$$

The prover and the verifier engage in the following protocol.

1. For $i = 1, \ldots, n$, the prover chooses a random $x_i \leftarrow \mathbb{Z}_q$, computes $U_i := B_i^{x_i}$, $V_i := K_i^{x_i}$, and $W_i := U_i^f$, and sends U_i, V_i, and W_i to the verifier.
2. The prover proves to the verifier

$$PK\{(x_1, \ldots, x_n, f) \; : \; B_1^{x_1} = U_1 \; \wedge \; K_1^{x_1} = V_1 \; \wedge \; \cdots \; \wedge \; B_n^{x_n} = U_n \; \wedge \; K_n^{x_n} = V_n \; \wedge \\ U_1^f = W_1 \; \wedge \; \cdots \; \wedge \; U_n^f = W_n \; \wedge \; B^f = K\}$$

3. For $i = 1, \ldots, n$, the verifier verifies that $V_i \neq W_i$.

Protocol 2 requires $6n + 1$ modular exponentiations for the prover and $6n + 2$ modular exponentiations for the verifier. Protocol 2 saves around 20% on performance compared with executing Protocol 1 for n times, as Protocol 2 only needs to prove $B^f = K$ once.

Protocol 3. Let p, q, and u be defined as in Protocol 1. The prover and the verifier have common input $B, K, B_I, K_1, \ldots, K_n \in \langle u \rangle$. The prover has the additional input f such that $B^f = K$. The prover wants to prove

$$PK\{(f) \; : \; B^f = K \; \wedge \; B_I^f \neq K_1 \; \wedge \; \cdots \; \wedge \; B_I^f \neq K_n\}$$

The prover and the verifier engage in the following protocol.

1. The prover chooses a random $x \leftarrow \mathbb{Z}_q$ and computes $U := B_I^x$ and $W := U^f$. For $i = 1, \ldots, n$, the prover computes $V_i := K_i^x$. The prover sends $U, V_1, \ldots, V_n, W$ to the verifier.
2. The prover proves to the verifier

$$PK\{(x, f) \; : \; B_I^x = U \; \wedge \; K_1^x = V_1 \; \wedge \; \cdots \; \wedge \; K_n^x = V_n \; \wedge \; U^f = W \; \wedge \; B^f = K\}$$

3. For $i = 1, \ldots, n$, the verifier verifies that $V_i \neq W$.

Protocol 3 requires $2n + 5$ modular exponentiations for the prover and $2n + 6$ modular exponentiations for the verifier. Protocol 3 saves around 73% on performance compared with executing Protocol 1 for n times, as Protocol 3 chooses the same x for different $K_1, \ldots, K_n$, thus it only needs to compute U and W once.

Same as Protocol 1, Protocol 2 and 3 are proof of knowledge protocols; but they are not zero-knowledge in the general case as the verifier cannot simulate the transcripts of the prover. However, we shall show in Section 6 that the signature generated by a group member (who runs Protocol 2 and 3 as the prover) can be simulated by the verifier under the decisional Diffie-Hellman assumption. The underlying idea is stated in the following claim.

Claim 1. *In Protocol 1, if $\log_B K$ is uniformly distributed in $\mathbb{Z}_q$ and $\log_{B_i} K_i$ is either uniformly distributed in $\mathbb{Z}_q$ or a value known to the verifier, then the transcripts of Protocol 1 can be simulated under the decisional Diffie-Hellman assumption.*

Proof. Let us denote $f = \log_B K$ and $f_i = \log_{B_i} K_i$. Lemma 1 states that given (U, V, W), transcripts of Protocol 1 can be simulated. We now describe how to simulate (U, V, W). The simulator chooses a random $a \leftarrow \mathbb{Z}_q$ and a random $y \leftarrow \langle u \rangle$ and set $U' = B_i^a$, $V' = K_i^a$, and $W' = y$. We argue that the verifier cannot distinguish the transcripts of Protocol 1 from the transcripts generated by the simulator, i.e.,

$$(U', V', W') \overset{c}{\approx} (U, V, W),$$

where $\overset{c}{\approx}$ stands for computationally indistinguishable.

In Protocol 1, $(U, V, W) = (B_i^x, K_i^x, B_i^{xf}) = (B_i^x, B_i^{xf_i}, B_i^{xf})$ for a random $x \in \mathbb{Z}_q$. Observe that the distributions of (U', V') and (U, V) are indistinguishable, as both a and x are chosen randomly from $\mathbb{Z}_q$. Also observe that $W = B_i^{xf}$ where x and f are chosen randomly in $\mathbb{Z}_q$, the distributions of (U', W') and (U, W) are computational indistinguishable under the decisional Diffie-Hellman assumption. For the same reason, if f_i is randomly chosen from $\mathbb{Z}_q$, then no computation-bounded adversary can distinguish (U, V) with (U, y) where $y \leftarrow \langle u \rangle$. Now, let us consider two cases. First case is that f_i also randomly distributed in $\mathbb{Z}_q$. Let C be a random variable equally distributed in $\langle u \rangle$, we have

$$(U, V, W) \overset{c}{\approx} (U, C, W) \overset{c}{\approx} (U', C, W') \overset{c}{\approx} (U', V', W').$$

The second case is that f_i already known to the verifier, we have

$$(U, V, W) = (U, U^{f_i}, W) \overset{c}{\approx} (U', U^{f_i}, W') = (U', V', W').$$

Therefore, the verifier cannot distinguish the distributions of (U, V, W) and (U', V', W'). $\square$

Same reasoning holds for Protocol 2 and 3. For Protocol 2, the simulator can, for $i = 1, \ldots, n$, simulate (U_i, V_i, W_i) as $(B_i^{a_i}, K_i^{a_i}, y_i)$ for $a_i \leftarrow \mathbb{Z}_q$ and $y_i \leftarrow \langle u \rangle$. For Protocol 3, the simulator can simulate $(U, V_1, \ldots, V_n, W)$ as $(B_I^a, K_1^a, \ldots, K_n^a, y)$ for $a \leftarrow \mathbb{Z}_q$ and $y \leftarrow \langle u \rangle$.

6 Security Proofs

Since the EPID scheme is built on top of the DAA scheme, the security proof of the EPID scheme comes largely from the DAA scheme [11] as well. To prove the security of our EPID scheme, we need to construct a simulator $\mathcal{S}$ such that the environment $\mathcal{E}$ cannot distinguish whether it runs in the real system, interacting with $\mathcal{A}$ and the real parties, or in the ideal system, interacting with $\mathcal{S}$ and the ideal parties. In this section, we first describe some assumption for the real system, then we construct a simulator, and in the end we prove that the simulator is formed correctly.

The main differences of our proof here with the proof of the DAA scheme are as follows:

1. We show that our proof of knowledge protocol for signature and issuer based revocations preserve the anonymity and unlinkability of the EPID scheme.

2. In the ideal model, the revocation manager can be either corrupted or uncorrupted, whereas the revocation manager in the DAA scheme is always corrupted.

3. In our scheme, every group member can be revoked. It follows that an honest group member can be revoked by the adversary using signature or issuer based revocation. In the DAA scheme, only corrupted users can be revoked.

6.1 Assumptions for Real System

To simplify the security proof of EPID, we make the following assumptions and modify the EPID protocols accordingly.

1. For signature based revocation in Section 4.6, we do not model the check of physical evidence when we revoke a prover based on his signature. Therefore, every group member can be revoked based on his signature by anybody in the system. In other words, in the security proof, the adversary can revoke an honest user based on the signature that the adversary has obtained from the user.

2. We assume every user at any given time has an updated revocation list. We do not consider the cases where the prover has one version of revocation list and the verifier has another. In the challenge phase in Section 4.5.1, the verifier sends only the message m and the nonce n_V. There is no need to send `sig-RL` from the verifier to the prover.

3. We do not model revocation manager and do not model signed revocation list. Everyone maintains his own `priv-RL`, `sig-RL`, and `issuer-RL` locally. Everybody has the ability to revoke and can act as a revocation manager. We assume that there is a broadcast channel so that the revocation messages are broadcast to everyone. We consider the following three revocations:

   (a) If a user finds a comprised private key (A, e, f, v), he broadcasts it to everyone. Everyone verifies the correctness of the private key, i.e., checks whether $A^e R^f S^v \equiv Z \pmod{N}$, and adds f to `priv-RL`.

   (b) If a user wants to revoke a group member based on the signature $\sigma = (\sigma_1, \sigma_2, \sigma_3)$ he obtained from the group member, he broadcasts σ to everyone. Everyone verifies the correctness of the signature and then adds (B, K) to `sig-RL`.

   (c) If the issuer wants to revoke a group member, he broadcasts (K, U, Σ) to everyone where the (K, U, Σ) tuple was obtained from the group member during the join protocol. Everyone verifies the correctness of Σ and then adds K to `issuer-RL`.

6.2 Simulator in the Ideal System

We now describe how to construct the simulator $\mathcal{S}$. Recall that $\mathcal{S}$ interacts with $\mathcal{T}$ on behalf of the corrupted parties of the ideal system, and simulates the real-system adversary $\mathcal{A}$ towards the environment $\mathcal{E}$. The simulator $\mathcal{S}$ is given $\mathcal{A}$ as a black box. The simulator $\mathcal{S}$ will use $\mathcal{A}$ to simulate the conversations of $\mathcal{E}$ with $\mathcal{A}$. That is, the simulator will forward all messages from $\mathcal{E}$ to $\mathcal{A}$ and simulates all messages from $\mathcal{A}$ to $\mathcal{E}$. As we are in the random oracle model, the simulator has full control of the random oracle, i.e., the simulator plays the random oracle towards $\mathcal{A}$. Whenever $\mathcal{S}$ gets a query from $\mathcal{A}$ to the random oracle, $\mathcal{S}$ has the ability to answer it in any way with the only constraint that it cannot give two different answers to the same query.

We now describe how $\mathcal{S}$ handles the different operations of the system. These operations are triggered either by requests from $\mathcal{T}$ to any of the corrupted party or by messages from $\mathcal{A}$ to any of the honest parties. The simulator $\mathcal{S}$ needs to handle the operations differently depending on which parties are corrupted. We name the cases as follows. A capital letter denotes that the corresponding party is not corrupted and a small letter denotes that is it corrupted. For instance, (Iu) denotes the case where the issuer is not corrupted, but the user is. If a party is not listed, then the simulator handles this case independently of whether or not this party is corrupted.

The simulator maintains three databases, a join database for recording information on each execution of the join protocol, a signature database for recording information on each signature, and a revocation database for recording information on each item in the revocation list. Each tuple in the databases has two entries: one is the information recorded, second is the user information. For example, if user $\mathcal{U}_j$ creates a signature σ, $\mathcal{S}$ inserts $(\sigma, \mathcal{U}_j)$ to the signature database. If the user is an unknown honest user, we use the keyword honest. Similarly, if the user is an unknonwn user controlled by the adversary, we use the keyword corrupted.

Ideal System Setup: For all parties controlled by $\mathcal{S}$, it indicates to $\mathcal{T}$ that they are corrupted.

Simulation of the Real System's Setup: There are two cases depending on whether the issuer is corrupted.

Case (i): The issuer is corrupted. $\mathcal{A}$ runs the setup program in the real system. $\mathcal{S}$ receives the issuer's public key $(N, g', g, h, R, S, Z, p, q, u)$ from $\mathcal{A}$ and then verifies the correctness of the public key. Note that in this case $\mathcal{S}$ does not know the factorization of N.

Case (I): The issuer is not corrupted. $\mathcal{S}$ runs the key generation of the issuer's and sends the thereby obtained public key $(N, g', g, h, R, S, Z, p, q, u)$ to $\mathcal{A}$ as the public key of the issuer. Note that in this case $\mathcal{S}$ knows the factorization of N.

Simulation of the Join Protocol: Players in this protocol are a user and the issuer.

Case (iu): Both the user and issuer are corrupted. There is nothing $\mathcal{S}$ has to do, i.e., it won't even notice as this is an internal transaction of $\mathcal{A}$.

Case (IU): Both the user and issuer are not corrupted. $\mathcal{S}$ does not have to do anything either. $\mathcal{S}$ won't even notice as this operation does not trigger a call from $\mathcal{T}$ to $\mathcal{S}$.

Case (Iu): The issuer is honest but the user is corrupted. In this case, $\mathcal{S}$ gets a request from $\mathcal{A}$ as real-system user $\mathcal{U}_j$ to run the join protocol. $\mathcal{S}$ plays the issuer and runs the join protocol with the adversary as the user as follows. $\mathcal{S}$ runs the join protocol until Step 6b, before sending (A, e, v'') out. If the protocol is aborted before this step, $\mathcal{S}$ does not need to do anything further for this query. Otherwise, $\mathcal{S}$ plays as the ideal-system user with $\mathcal{T}$ that the user wants to join. $\mathcal{S}$ waits until $\mathcal{T}$ tells $\mathcal{S}$ whether the user is allow to join. If the answer is positive, $\mathcal{S}$ finishes the join protocol with $\mathcal{A}$, otherwise it abort the join protocol. If the join protocol succeeds, $\mathcal{S}$ records transcript of the protocol along with $\mathcal{U}_j$ its join database.

Case (iU): The issuer is corrupted but not the user. In this case, $\mathcal{S}$ gets a request from $\mathcal{T}$ that the user $\mathcal{U}_j$ wants to join. Thus, $\mathcal{S}$ has to play the ideal-system issuer towards $\mathcal{T}$ and, in the same time, to simulate the real-system user towards $\mathcal{A}$. That is, $\mathcal{S}$ runs the real-system join protocol as the user with $\mathcal{A}$ as the issuer. If the join protocol finishes successfully and $\mathcal{S}$ obtains (A, e, v'') from $\mathcal{A}$, $\mathcal{S}$ stores the user's private key (A, e, f, v) and informs $\mathcal{T}$ that the user is allowed to join. $\mathcal{S}$ also stores transcript of the protocol along with $\mathcal{U}_j$ its join database. If the protocol fails, $\mathcal{S}$ informs $\mathcal{T}$ that the user is not allowed to join.

Simulation of the Proof of Membership Protocol: Players in this protocol are a prover and a verifier. We distinguish four cases, depending on whether or not the prover and verifier are corrupted. We assume that both the prover and verifier have the revocation lists `priv-RL` $= \{f_1, \ldots, f_{n_1}\}$, `sig-RL` $= \{(B_1, K_1), \ldots, (B_{n_2}, K_{n_2})\}$, and `issuer-RL` $= \{B_I, K_1, \ldots, K_{n_3}\}$.

Case (pv): Both the prover and verifier are corrupted. There is nothing $\mathcal{S}$ has to do, i.e., it won't even notice as this is basically an internal transaction of $\mathcal{A}$.

Case (PV): Both the prover and verifier are not corrupted. $\mathcal{S}$ does not have to do anything. Again, it won't even notice as this operation does not trigger a call from $\mathcal{T}$ to $\mathcal{S}$.

Case (pV): The prover is corrupted but not the verifier. In this case, $\mathcal{S}$ gets a request from $\mathcal{A}$ as a real-system prover to initiate a proof of membership protocol with a verifier. Thus $\mathcal{S}$ plays as the ideal-system prover towards $\mathcal{T}$ and, in the same time, to simulate the real-system verifier toward $\mathcal{A}$. First, $\mathcal{S}$ sends the request to $\mathcal{T}$ as an ideal-system prover. Later $\mathcal{T}$ responds with m to $\mathcal{S}$. $\mathcal{S}$ chooses n_V at random and sends m and n_V to $\mathcal{A}$ as the challenge. If $\mathcal{A}$ aborts, $\mathcal{S}$ informs $\mathcal{T}$ to abort. If $\mathcal{A}$ does not abort, $\mathcal{S}$ receives a signature σ from $\mathcal{A}$ on the message m. $\mathcal{S}$ first runs the verify protocol until Step 2. If the verification fails, $\mathcal{S}$ can just ignore the signature.

$\mathcal{S}$ then checks whether the signature comes from a revoked user in `priv-RL`, i.e., runs Step 3 of the verify protocol. If Step 3 fails, i.e., $\mathcal{S}$ finds some f_i in `priv-RL` such that $B^{f_i} = K$. If f_i is assigned to an honest user in $\mathcal{S}$'s revocation database and if the issuer is corrupted, $\mathcal{S}$ stops and outputs "failure 1". If f_i is assigned to an corrupted user $\mathcal{U}_j$, $\mathcal{S}$ stops and informs $\mathcal{T}$ that $\mathcal{U}_j$ wants to proceeds.

$\mathcal{S}$ next runs Step 4 of the verification process. If the verification fails and there exists no i such that $V_i = W_i$, $\mathcal{S}$ simply ignores the signature. If Step 4 fails because $V_i = W_i$ for some i. $\mathcal{S}$ looks up its revocation database and finds the revocation token (B_i, K_i) is assigned to user $\mathcal{U}_j$. If $\mathcal{U}_j$ is an honest user, $\mathcal{S}$ stops and outputs "failure 2". If $\mathcal{U}_j$ is a corrupted user, $\mathcal{S}$ stops and informs $\mathcal{T}$ that $\mathcal{U}_j$ wants to proceeds.

$\mathcal{S}$ then runs Step 5 of the verification process. If the verification fails and there exists no i such that $V_i = W$, $\mathcal{S}$ simply ignores the signature. If Step 5 fails because $V_i = W$ for some i. $\mathcal{S}$ looks up its revocation database and finds the revocation token K_i is assigned to user $\mathcal{U}_j$. If $\mathcal{U}_j$ is an honest user, $\mathcal{S}$ stops and outputs "failure 3". If $\mathcal{U}_j$ is a corrupted user, $\mathcal{S}$ stops and informs $\mathcal{T}$ that $\mathcal{U}_j$ wants to proceeds.

If the above verification succeeds, i.e., the signature from $\mathcal{A}$ is not revoked. So $\mathcal{S}$ has to figure out which user it should associate this signature. To this end, $\mathcal{S}$ checks whether it has already seen the (B, K) pair in σ appeared in the signature database. We have following cases:

1. If $\mathcal{S}$ used (B, K) as an honest user in the simulation of a signature, i.e., the (B, K) pair appears in the signature database and is associated with an honest user. $\mathcal{S}$ stops and outputs "failure 4".
2. If (B, K) was used by $\mathcal{A}$ before, $\mathcal{S}$ looks in its signature database to find which user $\mathcal{U}_j$ it assigned to that pair previously. Then $\mathcal{S}$ informs $\mathcal{T}$ to proceeds on behalf of $\mathcal{U}_j$. $\mathcal{S}$ inserts $(\sigma, \mathcal{U}_j)$ into its signature database.
3. If (B, K) is new to $\mathcal{S}$, $\mathcal{S}$ can just randomly select any corrupted $\mathcal{U}_j$ (no matter whether $\mathcal{U}_j$ has been selected before) such that $\mathcal{U}_j$ is not revoked. Then $\mathcal{S}$ informs $\mathcal{T}$ to proceed the proof of membership process as user $\mathcal{U}_j$. $\mathcal{S}$ records $(\sigma, \mathcal{U}_j)$ into its signature database.

Case (Pv): The prover is not corrupted but the verifier is. In this case, $\mathcal{S}$ obtains a request from $\mathcal{T}$ that some honest prover wants to perform a proof of membership. $\mathcal{S}$ plays the real-system prover and runs the proof of membership protocol with $\mathcal{A}$ as the real-system verifier as follows. $\mathcal{S}$ sends a request to $\mathcal{A}$ who responds with a message m and a nonce n_V. $\mathcal{S}$ sends m to $\mathcal{T}$ and waits until it gets a response from $\mathcal{T}$. If $\mathcal{T}$ aborts, $\mathcal{S}$ also aborts. Note that we assume that, in the ideal system, the honest prover proceeds only if the prover has not been revoked. Therefore, when $\mathcal{S}$ receives a notification from $\mathcal{T}$, the result must be that m has been signed by a non-revoked member. $\mathcal{S}$ proceeds as follows to simulate a signature in the real system (without knowing who is the prover and without knowing the prover's membership private key) using the power over the random oracle with $\mathcal{A}$ as the verifier as follows.

1. $\mathcal{S}$ picks a random $B \leftarrow \langle u \rangle$, then picks $T_1 \leftarrow \langle h \rangle$, $T_2 \leftarrow \langle g' \rangle$, and $K \leftarrow \langle B \rangle$.
2. $\mathcal{S}$ forges σ_1 with regard to B, K, T_1, and T_2 as follows:
   (a) $\mathcal{S}$ picks random integers
   $$\begin{aligned} s_v &\leftarrow \{0,1\}^{\ell_v+\ell_\varnothing+\ell_H}, & s_f &\leftarrow \{0,1\}^{\ell_f+\ell_\varnothing+\ell_H} \\ s_e &\leftarrow \{0,1\}^{\ell_{e'}+\ell_\varnothing+\ell_H}, & s_{ee} &\leftarrow \{0,1\}^{\ell_e+\ell_\varnothing+\ell_H+1}, \\ s_w, s_r &\leftarrow \{0,1\}^{\ell_N+2\ell_\varnothing+\ell_H}, & s_{ew}, s_{er} &\leftarrow \{0,1\}2^{\ell_e+\ell_N+2\ell_\varnothing+\ell_H+1}. \end{aligned}$$
   (b) $\mathcal{S}$ picks a random $c_1 \leftarrow \{0,1\}^{\ell_H}$.
   (c) $\mathcal{S}$ computes $s'_e := s_e + c_1 \cdot 2^{\ell_e}$ and
   $$\begin{aligned} \tilde{T}_1 &:= Z^{-c_1} T_1^{s'_e} R^{s_f} S^{s_v} h^{-s_{ew}} \bmod N, & \tilde{T}_2 &:= T_2^{-c_1} g^{s_w} h^{s'_e} (g')^{s_r} \bmod N, \\ \tilde{T}_3 &:= T_2^{-s'_e} g^{s_{ew}} h^{s_{ee}} (g')^{s_{er}} \bmod N, & \tilde{K} &:= K^{-c_1} B^{s_f} \bmod p. \end{aligned}$$
   (d) $\mathcal{S}$ patches the random oracle such that
   $$c_1 = H(N \| g' \| g \| h \| R \| S \| Z \| p \| q \| u \| B \| K \| T_1 \| T_2 \| \tilde{T}_1 \| \tilde{T}_2 \| \tilde{T}_3 \| \tilde{K} \| m \| n_V).$$
   (e) $\mathcal{S}$ sets $\sigma_1 := (B, K, T_1, T_2, c_1, s_v, s_f, s_e, s_r, s_w, s_{ew}, s_{ee}, s_{er})$.
3. $\mathcal{S}$ now forges σ_2 with regard to B and K as follows.
   (a) $\mathcal{S}$ chooses a random $s \leftarrow \mathbb{Z}_q$.
   (b) For $i = 1, \ldots, n_2$, $\mathcal{S}$ chooses
   $$x_i \leftarrow \mathbb{Z}_q, \qquad s_i \leftarrow \mathbb{Z}_q, \qquad W_i \leftarrow \langle u \rangle.$$
   and computes
   $$U_i := B_i^{x_i} \bmod p, \qquad V_i := K_i^{x_i} \bmod p.$$
   (c) $\mathcal{S}$ picks a random $c_2 \leftarrow \{0,1\}^{\ell_H}$.
   (d) $\mathcal{S}$ computes $\tilde{K} := K^{-c_2} B^s \bmod p$.
   (e) For $i = 1, \ldots, n_2$, $\mathcal{S}$ computes
   $$\tilde{U}_i := U_i^{-c_2} B_i^{s_i} \bmod p, \qquad \tilde{V}_i := V_i^{-c_2} K_i^{s_i} \bmod p, \qquad \tilde{W}_i := W_i^{-c_2} U_i^{s} \bmod p.$$
   (f) $\mathcal{S}$ patches the random oracle such that
   $$\begin{aligned} c_2 = H(&p \| q \| u \| B \| K \| \tilde{K} \| U_1 \| V_1 \| W_1 \| \tilde{U}_1 \| \tilde{V}_1 \| \tilde{W}_1 \| \ldots \\ &\| U_{n_2} \| V_{n_2} \| W_{n_2} \| \tilde{U}_{n_2} \| \tilde{V}_{n_2} \| \tilde{W}_{n_2} \| m \| \texttt{sig-RL} \| n_V). \end{aligned}$$
   (g) $\mathcal{S}$ sets $\sigma_2 := (B, K, c_2, s, U_1, V_1, W_1, s_1, \ldots, U_{n_2}, V_{n_2}, W_{n_2}, s_{n_2})$.
4. $\mathcal{S}$ now forges σ_3 with regard to B and K as follows.

(a) $\mathcal{S}$ first chooses

$$s_x \leftarrow \mathbb{Z}_q, \qquad s_f \leftarrow \mathbb{Z}_q, \qquad x \leftarrow \mathbb{Z}_q, \qquad W \leftarrow \langle u \rangle.$$

(b) $\mathcal{S}$ computes $U := B_I^x \bmod p$.

(c) For $i = 1, \ldots, n_3$, $\mathcal{S}$ computes $V_i := K_i^x \bmod p$.

(d) $\mathcal{S}$ picks a random $c_3 \leftarrow \{0,1\}^{\ell_H}$.

(e) $\mathcal{S}$ computes

$$\tilde{K} := K^{-c_3} B^{s_f} \bmod p, \quad \tilde{U} := U^{-c_3} B_I^{s_x} \bmod p, \quad \tilde{W} := W^{-c_3} U^{s_f} \bmod p.$$

(f) For $i = 1, \ldots, n_3$, $\mathcal{S}$ computes $\tilde{V}_i := V_i^{-c_3} K_i^{s_x} \bmod p$.

(g) $\mathcal{S}$ patches the random oracle such that

$$c_3 := H(p\|q\|u\|B\|K\|\tilde{K}\|U\|\tilde{U}\|V_1\|\tilde{V}_1\|\ldots\|V_{n_3}\|\tilde{V}_{n_3}\|W\|\tilde{W}\|m\|\texttt{issuer-RL}\|n_V).$$

(h) $\mathcal{S}$ sets $\sigma_3 := (B, K, c_3, s_x, s_f, U, V_1, \ldots, V_{n_3}, W)$.

5. $\mathcal{S}$ sends $\sigma := (\sigma_1, \sigma_2, \sigma_3)$ as signature on m to $\mathcal{A}$, then inserts $(\sigma, \mathsf{honest})$ into its signature database.

Simulation of Revocation: As we explained in Section 6.1, everyone can act as a revocation manager. We consider the following three revocations:

1. Private-key based Revocation. We consider the following two cases depending on whether the user is corrupted or not.

   Case (U). The user is not corrupted. If the issuer is corrupted, $\mathcal{S}$ cannot do anything here, as $\mathcal{S}$ does not know the factorization of N. If the issuer is not corrupted, $\mathcal{S}$ chooses a random (A, e, f, v) tuple such that $A^e R^f S^v \equiv Z \pmod{N}$, and sends (A, e, f, v) to $\mathcal{A}$. Note that the honest user cannot revoke any corrupted user using this method. $\mathcal{S}$ randomly picks up an uncorrupted user $\mathcal{U}_j$ and stores $(f, \mathcal{U}_j)$ in its revocation database.

   Case (u). The user is corrupted. $\mathcal{S}$ obtains (A, e, f, v) from $\mathcal{A}$. If $A^e R^f S^v$ is not equal to Z, $\mathcal{S}$ just ignores the values. Next, $\mathcal{S}$ checks whether f corresponds to a (B, K) pair that $\mathcal{S}$ used with an honest user during the simulation of signatures, i.e., whether there exists a (B, K) pair in the signature database such that $f = \log_B K$. If this is the case, $\mathcal{A}$ computed the discrete logarithm of K based on B, so $\mathcal{S}$ stops and outputs "failure 5". Next, we have the following two cases:

   - Let us first consider the case where the issuer is uncorrupted. $\mathcal{S}$ looks up its join database and checks whether there exists K in the database such that $f = \log_{B_I} K$. If $\mathcal{S}$ does not find any matching K, the adversary must forged a private key, $\mathcal{S}$ stops and outputs "failure 6". If $\mathcal{S}$ finds a K in its join database and associated with user $\mathcal{U}_j$, $\mathcal{S}$ stores $(f, \mathcal{U}_j)$ in its revocation database. Note that $\mathcal{U}_j$ must be a corrupted user, as in this case the issuer is uncorrupted and the join database only contains corrupted users.
   - The issuer is corrupted in this case. If there exists a K in $\mathcal{S}$'s join database such that $f = \log_{B_I} K$, $\mathcal{S}$ stops and outputs "failure 7". Otherwise, $\mathcal{S}$ picks a corrupted user $\mathcal{U}_j$ that has not been revoked and stores $(f, \mathcal{U}_j)$ in its revocation database. $\mathcal{S}$ also marks $\mathcal{U}_j$ as a revoked user.

2. Signature based revocation. We consider the following two cases depending on whether or not the user is corrupted.

   Case (U). The user is not corrupted. $\mathcal{S}$ chooses a signature from its signature database, i.e., either from the signatures it has simulated or from the signatures generated by $\mathcal{A}$. $\mathcal{S}$ sends the chosen signature to $\mathcal{A}$. $\mathcal{S}$ also inserts (B, K) in the signature to `sig-RL`. If the signature is simulated by $\mathcal{S}$, $\mathcal{S}$ randomly picks an honest (revoked or unrevoked) user $\mathcal{U}_j$, extracts (B, K) from the signature, and stores $(B, K, \mathcal{U}_j)$ in its revocation database. If the signature was generated by $\mathcal{A}$ before, $\mathcal{S}$ looks up its signature database and identify the signer $\mathcal{U}_j$, $\mathcal{S}$ extracts (B, K) from the signature and stores $(B, K, \mathcal{U}_j)$ in its revocation database. Note that whenever $\mathcal{S}$ adds a signature from $\mathcal{A}$ to its signature database, $\mathcal{S}$ is always able to associate the signature with a corrupted user.

   Case (u). The user is corrupted. $\mathcal{S}$ obtains a signature σ from $\mathcal{A}$. If the signature is not valid, $\mathcal{S}$ ignores the signatures. If (B, K) in σ already appears in the revocation list, $\mathcal{S}$ ignores the signatures. Otherwise, $\mathcal{S}$ adds (B, K) to `sig-RL`. $\mathcal{S}$ checks whether it has seen this signature before in its signature database. If not, $\mathcal{S}$ randomly picks a corrupted user $\mathcal{U}_j$ and stores $(B, K, \mathcal{U}_j)$ in its revocation database. If σ already exists in the signature database and is tagged with `honest`, i.e., the signature was simulated by $\mathcal{S}$, $\mathcal{S}$ randomly picks an honest user $\mathcal{U}_j$ and stores $(B, K, \mathcal{U}_j)$ in its revocation database. If σ exists in the signature database and is associated with a corrupted user $\mathcal{U}_j$, $\mathcal{S}$ stores $(B, K, \mathcal{U}_j)$ in its revocation database.

3. Issuer based revocation based. We consider the following two cases depending on whether or not the issuer is corrupted.

   Case (I). The issuer is not corrupted. $\mathcal{S}$ either chooses a (K, U, Σ) tuple from its join database or creates a new (K, U, Σ) tuple. If the (K, U, Σ) tuple comes from $\mathcal{S}$'s join database, $\mathcal{S}$ identifies the user $\mathcal{U}_j$ associated with that tuple. Note that in this case $\mathcal{U}_j$ must be a corrupted user. $\mathcal{S}$ inserts $(K, \mathcal{U}_j)$ into its revocation database. Otherwise, $\mathcal{S}$ randomly chooses an uncorrupted user $\mathcal{U}_j$, runs the join protocol locally, and obtains the tuple (K, U, Σ) from the transcript of the join protocol. $\mathcal{S}$ then inserts $(K, \mathcal{U}_j)$ into its revocation database. In the end, $\mathcal{S}$ adds K to `issuer-RL` and sends (K, U, Σ) to $\mathcal{A}$.

   Case (i). The issuer is corrupted. $\mathcal{S}$ obtains a (K, U, Σ) tuple from $\mathcal{A}$. If the proof of knowledge Σ is not valid, $\mathcal{S}$ ignores this tuple. If K already appears in the issuer based revocation list, $\mathcal{S}$ ignores the tuple. Otherwise, $\mathcal{S}$ adds K to `issuer-RL`. $\mathcal{S}$ looks up its join database and check whether (K, U, Σ) exists in the database. If (K, U, Σ) exists in the join database and corresponds to honest user $\mathcal{U}_j$, $\mathcal{S}$ inserts $(K, \mathcal{U}_j)$ in its revocation database. Otherwise, $\mathcal{S}$ randomly picks a corrupted user that has not been revoked in `priv-RL`, and adds $(K, \mathcal{U}_j)$ in its revocation database.

This concludes the description of the simulator $\mathcal{S}$. What remains to argue is that the environment cannot distinguish whether it runs in the ideal system or in the real system.

6.3 Correctness of the Simulator

We now argue that the simulator described above works. That is, under the decisional Diffie-Hellman assumption and the strong RSA assumption, the simulator will not stop and output "failure" and that the environment cannot distinguish whether or not it is run in the real system or the ideal system. We next discuss each failure case in details.

- Failure 1: This failure only occurs if the issuer is not corrupted. If this failure occurs, the adversary has forged a revoked signature with respect to f_i that the simulator chosen. Using the rewinding techniques, we can extract (A, e, f_i, v) from this signature. We shall show in Lemma 4 that this is not possible under the strong RSA assumption.

- Failure 2: If this failure occurs, the adversary has forged a signature with respect to (B, K) such that $\log_B K = \log_{B_i} K_i$ where (B_i, K_i) comes from an honest user, i.e., is chosen by the simulator. Since the signature is based on a proof of knowledge of the discrete logarithm of K, we can extract $\log_B K$ and $\log_{B_i} K_i$ using rewinding techniques on the adversary and using the power over the random oracle. Thus, we can reduce an adversary that produce a failure to this type to one that computes discrete logarithms. Note that such a reduction would loose a factor because of the oracle calls.

- Failure 3: If this failure occurs, the adversary has forged a signature with respect to (B, K) such that $\log_B K = \log_{B_I} K_i$ where K_i comes from an honest user during the join protocol, i.e., is chosen by the simulator. Since the signature is based on a proof of knowledge of the discrete logarithm of K, we can extract $\log_B K$ and $\log_{B_I} K_i$ using rewinding techniques on the adversary and using the power over the random oracle. Also observe that the simulator can choose any B_I he wants as he controls the random orale. Thus, we can reduce an adversary that produce a failure to this type to one that computes discrete logarithms. Same as failure 2, such a reduction would loose a factor because of the oracle calls.

- Failure 4: If this failure occurs, the adversary has forged a signature respect to (B, K), which appeared in a signature simulated by $\mathcal{S}$. Similar to failure 1, we can extract $\log_B K$ using rewinding on the adversary and using the power over the random oracle. Thus, we can reduce an adversary that produce a failure to this type to one that computes discrete logarithms.

- Failure 5: The adversary has produced an f such that B^f equals K for some (B, K). However, the simulator choose all the K randomly without even knowing $\log_B K$. It is straightforward to show that the adversary could be used to solve the discrete logarithm problem in $\mathbb{Z}_p^*$.

- Failure 6: This is a similar failure as failure 1 and can only occur if the issuer is honest. The adversary has forged a private key (A, e, f, v) such that f does not correspond any K in the simulator's join database. In other words, the adversary did not obtain a private key with respect to f from the issuer, but forged a valid membership private key. We shall show in Lemma 4 that this is not possible under the strong RSA assumption.

- Failure 7: This failure only occurs if the issuer is corrupted. The simulator ran the join protocol with the adversary as the real-system issuer. The simulator has chosen a random f and revealed (B, K) to the adversary such that B^f equals K. Later, the adversary outputs (A, e, f, v) such that f is equal to $\log_B K$. Using the power over the random oracle, given a (B, K) pair, the adversary can compute f such that $B^f = K$. Thus, we can reduce an adversary that produce a failure to this type to one that computes discrete logarithms.

It remains to argue that the environment and adversary cannot distinguish whether they run in the real system or in the ideal system. Observe that, except for the join protocol, the simulator behaves exactly the same as the honest players in the real system. In Lemma 3, we shall show that signatures generated by an honest user can be simulated by the simulator. Therefore, our EPID scheme is secure.

Lemma 3. *Under the decisional Diffie-Hellman assumption, there exists no adversary can distinguish a signature produced by an honest user in the real system from a signature generated by the simulator in the ideal system.*

Proof. To simulate a signature from an honest user, the simulator first simulates the first part of the signature $\sigma_1 = (B, K, T_1, T_2, c_1, s_v, s_f, s_e, s_r, s_w, s_{ew}, s_{ee}, s_{er})$, then simulates the second part of the signature $\sigma_2 = (B, K, c_2, s, U_1, V_1, W_1, s_1, \ldots, U_{n_2}, V_{n_2}, W_{n_2}, s_{n_2})$, finally simulates the third part of the signature $\sigma_3 = (B, K, c_3, s_x, s_f, U, V_1, \ldots, V_{n_3}, W)$.

As in the DAA scheme [11], σ_1 is correctly simulated. As the simulator controls the random oracle and patches the random oracle such that its outputs are uniformly random, the distribution of c_1 is the same in both the real system and the ideal system. It is easy to see that s_e, s_r, s_w, s_{ew}, s_{ee}, and s_{er} are distributed statistically chose in both the real system and the ideal system if $\ell_\varnothing$ is sufficiently large. For value B, both the user and simulator choose B at random. Next consider the the value K, the simulator chooses K at random whereas in the real system $\log_B K$ is always the same for a given user. Under the decisional Diffie-Hellman assumption, no adversary can distinguish two distributions. Next consider the values T_1 and T_2. The simulator chooses them randomly from $\langle h \rangle$ and $\langle g' \rangle$, respectively. An honest user on the other hand computes T_1 as Ah^w and T_2 as $g^w h^e (g')^r$. Observe that if $A \in \langle h \rangle$ and $g, h \in \langle g' \rangle$, T_1 and T_2 are distributed statistically close to random elements of $\langle h \rangle$ and $\langle g' \rangle$. The issuer already proves that $A \in \langle h \rangle$ in the join protocol and $g, h \in \langle g' \rangle$ in the setup process.

We now show that σ_2 is correctly simulated. Note that the values of B and K are the same as in σ_1. Since the simulator controls the random oracle, c_2 is randomly distributed in both the ideal and real system. It is also easy to see that $s, s_1, \ldots, s_{n_2}$ are distributed exactly the same in both cases. We now need to consider the distributions of (U_i, V_i, W_i) for $i = 1, \ldots, n_2$. Note that for each (B_i, K_i) pair in `sig-RL`, if it is from an honest user, then f_i is randomly chosen by the simulator. If it is from the adversary, then f_i is chosen arbitrarily by the adversary. Based on Claim 1, the adversary cannot distinguish the distributions of (U_i, V_i, W_i) generated by the simulator in the ideal system from the distributions generated by an honest user in the real system.

We now show that σ_3 is correctly simulated. Note that the values of B and K are the same as in σ_1. Again, since the simulator controls the random oracle, c_3 is randomly distributed in both the ideal and real system. It is also easy to see that s_x and s_f are distributed exactly the same in both cases. We now need to consider the distributions of $(U, V_1, \ldots, V_{n_3}, W)$. Note that for each K_i in `sig-RL`, if it is from an honest user, then $f_i = \log_{B_I} K_i$ is randomly chosen by the simulator and is equally distributed over $\mathbb{Z}_q$. If it is from the adversary, then f_i is chosen arbitrarily by the adversary. Based on the similar argument as in Claim 1, the adversary cannot distinguish the distributions of $(U, V_1, \ldots, V_{n_3}, W)$ generated by the simulator in the ideal system from the distributions generated by an honest user in the real system under the decisional Diffie-Hellman assumption. □

Lemma 4. *Under the strong RSA assumption, there exists no adversary who does not control the issuer but can make the simulator output failure 1 or failure 6 in the above simulation, providing that the join protocol is executed sequentially.*

Proof. (sketch) This proof is essentially same as the proof of Lemma 3 in [11]. If there exists an adversary that makes the simulator output failure 1 or 6, then this adversary can forge a new membership private key (A, e, f, v) that has not been issued. Since all the protocols in the EPID scheme are proof of knowledge protocols, we can rewind the adversary to output a CL signature on f. Given that the CL signature scheme [14] is secure under the strong RSA assumption; i.e., no adversary can forge a CL signature, this Lemma holds. □

7 Using EPID with Tamper Resistant Hardware

As in many group signature schemes (e.g., [1, 4, 18, 25], to list a few), the EPID scheme can be used without any hardware support, i.e., all the protocols in EPID are run on software. However, if the EPID scheme is used in combine with tamper resistant hardware such as TPM and smartcard, we could have the following benefits: (1) less computational work for trusted hardware device, and (2) more efficient revocation mechanism.

Suppose the EPID scheme is used with support of trusted tamper resistant hardware, we can use the trusted hardware for storing the user's membership private key and for performing proof of membership protocol. As in the DAA scheme [10], the computation of the proof of membership protocol can be partially outsourced to the host in which the hardware device is embedded. We omit the details on how to do such outsourcing and refer the readers to [10]. The main design principle is that the host and the hardware joinly perform the proof of membership as the prover. The host, if corrupted, could break the anonymity of the user but cannot learn the user's membership private key. Because in any case, the host can pad some identifier to each message sent by the hardware device.

Another advantage of using trusted hardware device for EPID is to have more efficient revocation. In EPID scheme, we could have the following five cases where we want to revoke a user.

1. The user's membership private key was removed from the trusted hardware device, and was published widely so that everyone knows this compromised private key.

2. The user's membership private key was extracted from the trusted hardware device by the adversary. The issuer suspects that the user's hardware device was compromised, but has not obtained the user's private key.

3. The user's membership private key was extracted form the hardware device by the adversary. The revocation manager suspects that the hardware device was corrupted. The revocation manager obtains a signature from the corrupted device but has not obtained the private key.

4. The issuer revokes the user for some management reason, e.g., the user left the group or the user's group membership expired.

5. The user is revoked from transactions. More specifically, the user abuses his group privilege and is revoked by the revocation manager after the user conducted a proof of membership.

To handle the above revocations using the EPID scheme in Section 4, we can use private-key based revocation to revoke users of case 1, issuer based revocation to revoke users of cases 2 and 4, and signature based revocation to revoke users of cases 3 and 5. Note that, we can categorize the above revocations into two types:

- The hardware device that contains the membership private key was compromised or suspected to be compromised by the adversary. This type corresponds to cases 1-3 in the above list.

- The user of the hardware device needs to be revoked while the hardware device held by the user is not suspected of being compromised. In other words, the hardware device remains uncontrolled by the adversary. This type corresponds to cases 4 and 5 in the above list.

We can modify our EPID scheme to have more efficient revocation as follows. For each of the above five cases, the revocation manager maintains a revocation list. For case 1, we still use private-key based revocation, i.e., the verifier checks against the revocation list locally. For case

2, we use issuer based revocation, i.e., the user needs to prove that his private key is not in the revocation list of this case. For case 3, we use signature based revocation, i.e., the user needs to prove that his private key is not in the revocation list of this type. For cases 4 and 5, instead of conducting expensive proof of knowledge, the hardware device can check its private key against the corresponding revocation lists. If the hardware device determines that it has not been revoked, it produces an EPID signature, stating that it was not in the revocation lists of cases 4 and 5. This new revocation method is correct because we assume that the hardware device in cases 4 and 5 is not compromised, even though the user of the device has been revoked or compromised. If the device indeed contained the private key that has been revoked in the revocation lists for cases 4 and 5, then the device would not sign any such statement.

More specifically, let $\{K_1, \ldots, K_{n4}\}$ be the revocation list of case 4. The hardware device only needs one modulo exponentiation, i.e., it computes $K := B_I^f \bmod p$. The hardware device then compares K against the revocation list to make sure $K \notin \{K_1, \ldots, K_{n4}\}$. Similiarly, let $\{(B_1, K_1), \ldots, (B_{n5}, K_{n5})\}$ be the revocation list of case 5. For each item (B_i, K_i) the revocation list of case 5, the hardware device only needs one modulo exponentiation. It simply computes $K := B_i^f \bmod p$ and verifies that $K \neq K_i$.

In practice, we believe that most of revocations belong to case 4. Take the driver's license example, most of revocations happen because the Department of Motor Vehicles wants to revoke a user or because a license has been expired, but not because a tamper resistant device used for driver's license has been broken. Consider that the revocation method is very efficient for case 4, using EPID with tamper resistant hardware becomes an atttractive solution.

8 Conclusion

We described the notion of EPID and gave an efficient construction to the EPID scheme based the strong RSA assumption and the decisional Diffie-Hellman assumption. To prove membership, both the prover and verifier need to perform computations linear to the size of the revocation list. One future direction is to develop more efficient revocation methods, i.e., revocation requires only sub-linear work for the prover or the verifier. Another possible extension to the EPID scheme is to improve the join protocol in such a way that the issuer could run the join protocol concurrently with different users.

References

[1] Giuseppe Ateniese, Jan Camenisch, Marc Joye, and Gene Tsudik. A practical and provably secure coalition-resistant group signature scheme. In *Advances in Cryptology — CRYPTO '00*, volume 1880 of *LNCS*, pages 255–270. Springer, 2000.

[2] Giuseppe Ateniese, Dawn Xiaodong Song, and Gene Tsudik. Quasi-efficient revocation in group signatures. In *Proceedings of the 6th International Conference on Financial Cryptography*, volume 2357 of *LNCS*, pages 183–197. Springer, 2002.

[3] Mihir Bellare, Juan A. Garay, and Tal Rabin. Fast batch verification for modular exponentiation and digital signatures. In *Advances in Cryptology — EUROCRYPT '98*, volume 1403 of *LNCS*, pages 236–250. Springer, 1998.

[4] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *Advances in Cryptology — CRYPTO '04*, volume 3152 of *LNCS*, pages 41–55. Springer, 2004.

[5] Dan Boneh and Hovav Shacham. Group signatures with verifier-local revocation. In *Proceedings of 11th ACM Conference on Computer and Communications Security*, pages 168–177, October 2004.

[6] Fabrice Boudot. Efficient proofs that a committed number lies in an interval. In *Advances in Cryptology — EUROCRYPT '00*, volume 1807 of *LNCS*, pages 431–444, May 2000.

[7] Stefan A. Brands. *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy*. MIT Press, August 2000.

[8] Emmanuel Bresson and Jacques Stern. Efficient revocation in group signatures. In *Proceedings of the 4th International Workshop on Practice and Theory in Public Key Cryptography*, pages 190–206. Springer, 2001.

[9] Ernest F. Brickell, David Chaum, Ivan Damgård, and Jeroen van de Graaf. Gradual and verifiable release of a secret. In *Advances in Cryptology — CRYPTO '87*, volume 293 of *LNCS*, pages 156–166. Springer, 1987.

[10] Ernie Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. In *Proceedings of the 11th ACM Conference on Computer and Communications Security*, pages 132–145. ACM Press, 2004.

[11] Ernie Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. Cryptology ePrint Archive, Report 2004/205, 2004. `http://eprint.iacr.org/`.

[12] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *Advances in Cryptology — EUROCRYPT '01*, volume 2045 of *LNCS*, pages 93–118. Springer, 2001.

[13] Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *Advances in Cryptology — CRYPTO '02*, volume 2442 of *LNCS*, pages 61–76. Springer, 2002.

[14] Jan Camenisch and Anna Lysyanskaya. A signature scheme with efficient protocols. In *Proceedings of the 3rd Conference on Security in Communication Networks*, volume 2576 of *LNCS*, pages 268–289. Springer, 2002.

[15] Jan Camenisch and Markus Michels. Proving in zero-knowledge that a number is the product of two safe primes. In *In Advances in Cryptology — EUROCRYPT '99*, volume 1592 of *LNCS*, pages 106–121. Springer, 1999.

[16] Jan Camenisch and Markus Michels. Separability and efficiency for generic group signature schemes. In *In Advances in Cryptology — CRYPTO '99*, volume 1666 of *LNCS*, pages 413–430. Springer, 1999.

[17] Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *Advances in Cryptology — CRYPTO '03*, volume 2729 of *LNCS*, pages 126–144. Springer, 2003.

[18] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups. In *Advances in Cryptology — CRYPTO '97*, volume 1296 of *LNCS*, pages 410–424. Springer, 1997.

[19] Ran Canetti. *Studies in Secure Multiparty Computation and Applications.* PhD thesis, Weizmann Institute of Science, Rehovot, Israel, 1995.

[20] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

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

[22] David Chaum. Zero-knowledge undeniable signatures. In *Advances in Cryptology — EUROCRYPT '90*, volume 473 of *LNCS*, pages 458–464. Springer, 1990.

[23] David Chaum, Jan-Hendrik Evertse, and Jeroen van de Graaf. An improved protocol for demonstrating possession of discrete logarithms and some generalizations. In *Advances in Cryptology — EUROCRYPT '87*, volume 304 of *LNCS*, pages 127–141. Springer, 1987.

[24] David Chaum and Torben P. Pedersen. Wallet databases with observers. In *Advances in Cryptology — CRYPTO '92*, volume 740 of *LNCS*, pages 89–105. Springer, 1992.

[25] David Chaum and Eugène van Heyst. Group signatures. In *Advances in Cryptology — EUROCRYPT '91*, volume 547 of *LNCS*, pages 257–265. Springer, 1991.

[26] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Advances in Cryptology — EUROCRYPT '94*, volume 839 of *LNCS*, pages 174–187. Springer, 1994.

[27] Ivan Damgård and Eiichiro Fujisaki. An integer commitment scheme based on groups with hidden order. In *Advances in Cryptology — ASIACRYPT '02*, volume 2501 of *LNCS*, pages 125–142. Springer, December 2002.

[28] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology — CRYPTO '86*, volume 263 of *LNCS*, pages 186–194. Springer, 1987.

[29] Eiichiro Fujisaki and Tatsuaki Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In *Advances in Cryptology — CRYPTO '97*, volume 1294 of *LNCS*, pages 16–30. Springer, 1997.

[30] Joe Kilian and Erez Petrank. Identity escrow. In *Advances in Cryptology — CRYPTO '98*, volume 1642 of *LNCS*, pages 169–185. Springer, 1998.

[31] Arjen K. Lenstra and Eric R. Verheul. Selecting cryptographic key sizes. *Journal of Cryptology*, 14(4):255–293, 2001.

[32] Birgit Pfitzmann and Michael Waidner. Composition and integrity preservation of secure reactive systems. In *Proceedings of 7th ACM Conference on Computer and Communications Security*, pages 245–254, November 2000.

[33] Birgit Pfitzmann and Michael Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 184–200. IEEE Computer Society Press, 2001.

[34] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In *Advances in Cryptology — EUROCRYPT '96*, volume 1070 of *LNCS*, pages 387–398. Springer, 1996.

[35] Claus P. Schnorr. Efficient identification and signatures for smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[36] Dawn Xiaodong Song. Practical forward secure group signature schemes. In *Proceedings of the 8th ACM Conference on Computer and Communications Security*, pages 225–234. ACM Press, 2001.

[37] Trusted Computing Group. TCG TPM specification 1.2, 2003. Available at `http://www.trustedcomputinggroup.org`.

[38] Trusted Computing Group website. `http://www.trustedcomputinggroup.org`.