Privacy-Preserving Cryptography

Fabrice Mouhartem

1 Privacy Preserving Cryptography

Privacy-preserving cryptography is a field of cryptography that aims at providing methods to ensure some functionality while preserving the anonymity of the participants. An example of such method could be *anonymous credentials* [11, 7]. In this primitive, a user should be able to authenticate to some service without revealing its identity nor the pattern of the authentications. It can be used for access control as in proving the right to access some buildings while keeping individual access schedule private. This system involves one (or more) credential issuer(s) and a set of users who have their own secret keys and pseudonyms that are bound to their secret. Users can dynamically obtain credentials from an issuer that only knows users' pseudonyms ad obliviously sign users' secret key as well as a set of attributes. Later on, users can make themselves known to verifiers under a different pseudonym and demonstrate possession of a certification from the issuer, without revealing neither their signature nor the secret key.

More examples exist: some related to anonymous authentication such as group signatures, ring signatures, blind signatures, or more intricate tasks such as electronic cash or electronic voting.

Contents

2 Group Signatures

2.1 Definition

In this section we will use the example of group signatures to work on privacy-based primitives.

Group signatures was coined by Chaum and Van Heyst [12] and later formalised by Bellare Micciancio and Warinschi [2] in the *static* setting where the group is defined *once-and-for-all* at the setup phase. This model was later extended for dynamically growing groups concurrently by Bellare, Shi and Zhang and Kiayias and Yung [3, 19].

A dynamic group signature is a protocol enabling users to sign a message on behalf of a group it priorly enrolled while remaining anonymous inside this group. Moreover, in case of conflicts, an opening authority is able to lift anonymity of problematic signatures using its own secret information. As an application, we can consider anonymous access control as in public transportation. Indeed, a public transport company only needs to check whether a user possesses a valid subscription or not. Thus, when a user subscribes to the public transport system, it joins the group. Later on, when a user commutes, it signs a challenge, for instance the timestamp of entry, and the correct verification of the signature means that it is allowed to travel. In case of problem, it is then possible to a third party, for instance law enforcement, to deanonymise signatures to know who was in the public transport at this time.

To begin with, we will consider the case of static groups.

Definition 1 (Group Signatures)**.** A (static) group signature is a tuple of 4 algorithms $(\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Open})$ acting as follows:

Setup$(1^\lambda, N)$**:** this algorithm takes as inputs the security parameter λ and the number of users N and outputs the group public key gpk, secret keys $(\mathsf{gsk}[i])_{i=1}^N$ and the opening key ok.

Sign$(\mathsf{gsk}[i], m)$**:** it takes as inputs a user secret key $\mathsf{gsk}[i]$, a message m to be signed and outputs a signature σ.

Verify$(\mathsf{gpk}, m, \sigma)$**:** given a message m, a signature σ and the group public key gpk, this algorithm returns true or false.

Open(ok, m, σ)**:** given the opening key ok, a message m and a signature σ, this algorithm returns an identity i or an error message $\perp$.

These algorithms act as their names suggest it. We now need to define the security notions that are wanted for this constructions. Bellare, Shi and Zhang and Kiayias and Yung [3, 19] boiled down the security of the scheme to the following properties beside correctness: firstly the identity of the users is protected by *anonymity*, which states that without the opening key, it is impossible to tell who provided a valid signature. Then, there is the *traceability*, which is the security of the system against malicious users even if they colludes with the authorities. In other words, it is impossible to provide a valid signature that cannot be traced back by the opening algorithm. More formally, theses notions are defined as follows.

Correctness. A group signature is said to be *correct* if for any integer N, any $\mathsf{gpk}, (\mathsf{gsk}[i])_{i=1}^N$ obtained from the Setup algorithm, any message m and index $i \in [1, N]$, it holds that

$$\mathsf{Verify}(\mathsf{gpk}, \mathsf{Sign}(\mathsf{gsk}[i], m)) = \mathsf{true}$$

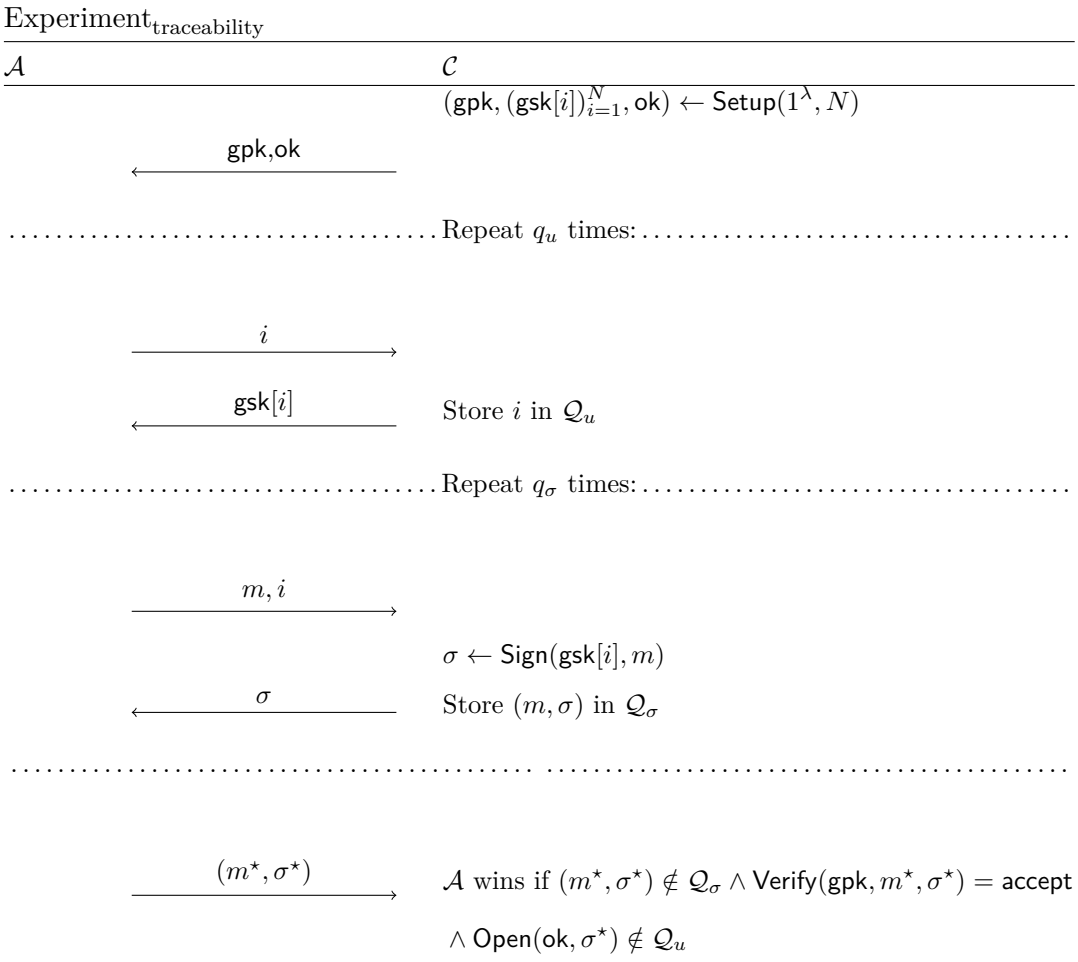

Figure 1: The traceability game

Traceability. A group signature is *traceable* if no probabilistic polynomial-time (PPT) algorithm can win the traceability game described in Figure 1 with non-negligible probabilities. This winning probability is also denoted by advantage.

(Full-)Anonymity. A group signature is *fully-anonymous* if for all PPT algorithms, the following advantage with respect to the anonymity game in Figure 2 is negligible where $\mathsf{Adv}^{\text{anon}}(\lambda) \triangleq \left|\Pr[d = 1 \mid \text{Experiment}^{1}_{\text{anon}}] - \Pr[d = 1 \mid \text{Experiment}^{0}_{\text{anon}}]\right|$.

2.2 Non-Interactive Zero-Knowledge Proofs

Alongside with a formal definition for group signatures, Bellare, Micciancio and Warinschi proposed a generic construction for group signatures that rely on an existentially unforgeable signature scheme, a public-key encryption scheme and non-interactive zero-knowledge proofs.

Given an NP-language $\mathcal{L} = \{(x, w)\}$, a non-interactive zero-knowledge proofs enables proving statements on $(x, w) \in \mathcal{L}$ being given x.

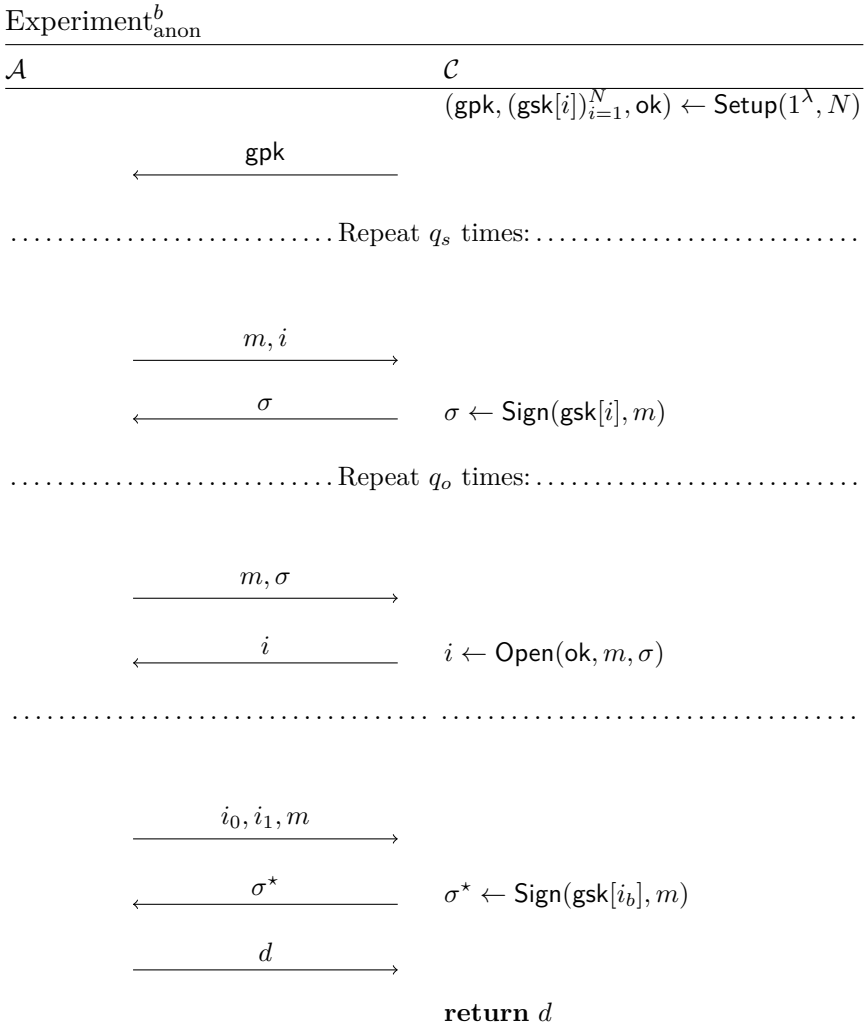

Figure 2: The anonymity game

Definition 2 (Non-Interactive Zero-Knowledge Proofs)**.** A non-interactive zero-knowledge (NIZK) proof system on an NP language $\mathcal{L}$ is a tuple of algorithms (Setup, Prove, Verify) such that:

Setup(1^{λ})**:** Given a security parameter λ, this algorithm returns a common reference string crs.

Prove(crs, x, w)**:** From an element x and its corresponding witness w, output a proof π.

Verify(crs, x, π)**:** This algorithm inputs an element x and a proof π and outputs either accept or reject.

The security of NIZK proofs is defined using the correctness notion of *completeness*, the *soundness*, which is the security of the verifier against dishonest provers who want to provide a proof π without knowing a witness w, and the *zero-knowledgeness*, which on the other hand models the security of the prover which wants no information to be leaked besides the validity of the statement against (potentially) malicious verifiers.

Completeness. A non-interactive proof system on language $\mathcal{L}$ is *complete* if for any $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$, any $(x, w) \in \mathcal{L}$ it holds that $\mathsf{Verify}(\mathsf{crs}, x, \mathsf{Prove}(\mathsf{crs}, x, w)) = \mathsf{accept}$

Soundness. A non-interactive proof system on language $\mathcal{L}$ is *sound* if for any security parameter λ, any (possibly corrupted or unbounded) prover $\widehat{\mathsf{P}}$, for any $x \notin \mathcal{L}$, the following probability is negligible in λ:

$$\Pr\left[\mathsf{crs} \hookleftarrow \mathcal{U}(\{0,1\}^{p(\lambda)}); \pi \leftarrow \widehat{\mathsf{P}}(\mathsf{crs}, x); \mathsf{Verify}(\mathsf{crs}, \pi, x) = \mathsf{accept}\right].$$

In the case where $\widehat{\mathsf{P}}$ is restricted to be a PPT algorithm, we say that we have an *argument* of knowledge.

Zero-Knowledge. A non-interactive proof system on language $\mathcal{L}$ is *zero-knowledge* if for any security parameter λ, there exists a simulator $\mathsf{Sim} = (\mathsf{S}_1, \mathsf{S}_2)$ such that S_1 outputs a $\mathsf{crs} \in \{0,1\}^{p(\lambda)}$ and a state information τ_{crs} on input λ, and S_2 outputs a proof π from τ_{crs} and x. Given $(x, w) \in \mathcal{L}$ the two following distributions are computationally indistinguishable:

$$\left\{(\mathsf{crs}, \pi) \mid (\mathsf{crs}, \tau_{\mathsf{crs}}) \leftarrow \mathsf{S}_1(\lambda), \pi \leftarrow \mathsf{S}_2(\tau_{\mathsf{crs}}, x)\right\}$$

and

$$\left\{(\mathsf{crs}, \pi) \mid \mathsf{crs} \hookleftarrow \mathcal{U}(\{0,1\}^{p(\lambda)}), \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, w)\right\}.$$

Moreover, for the security proof to hold in the case of the construction of group signatures we will consider, the following strengthening on NIZK proofs is required:

Simulation Soundness. A NIZK proof system is *simulation sound* if the soundness is verifier even under the view of simulated proofs.

Remark. Let us notice that without a crs, it is impossible to have NIZK for all NP. Indeed, let us assume that there exist a NIZK proof system (P, V) for a language $\mathcal{L} \in \mathsf{NP}$. Then we will see that it implies $\mathcal{L} \in \mathsf{BPP}$.

To do this, let us construct a PPT algorithm $\mathcal{A}$ that decides if $x \in \{0,1\}^\star$ is in $\mathcal{L}$ or not. From the zero-knowledge property, we know that there is a PPT simulator that computes $\pi \hookleftarrow \mathsf{Sim}(1^\lambda, x)$ such that $V(\pi, x) = 1$ with overwhelming probability (as it is distributed as a real proof). Moreover, from the soundness property, we know that the probability that $V(\pi, x) = 1$ if $x \notin \mathcal{L}$ is negligibly small. Therefore, $\mathcal{A}$ can simply run Sim on input x and check whether the proof is approved by the verifier or not, giving a probabilistic algorithm to decide if $x \in \mathcal{L}$.

We can also notice that in the presence of a crs, this proof does not hold, as it implicitly means that the simulator chooses the crs on the fly. However, in the presence of a crs, the soundness holds with respect to a *fixed* crs that is chosen beforehand (that the cheating prover cannot change). Hence the necessity either to have a crs or more generally interactions to have ZK that doesn't rely on set-up assumptions.

To construct NIZK arguments, we will use the Fiat-Shamir heuristic. This transformation uses a Σ-protocol and a hash function modeled as a random oracle.

Definition 3 (Σ-protocol)**.** A Σ-*protocol* (P, V) over an NP-language $\mathcal{L}$ is a 3-round protocol (a, c, z) between a prover P possessing $(x, w) \in \mathcal{L}$ and a verifier V knowing only x. The interaction starts with the prover sending a message to the verifier. The protocol should verify the following security properties:

- Completeness: if the protocol is followed honestly between a prover and a verifier, then the verifier should accept the proof.
- Special soundness: given two accepting different transcripts with the same commitment $a : (a, c, z)$ and (a, c', z'), there exists an extractor that efficiently computes w for the x used in the proof.
- Honest-verifier zero-knowledge: there exists a simulator that on inputs x and a random challenge c, outputs a transcript (a, c, z) with the same distribution as a real interaction.

Using such a protocol, it is possible to generically transform it into a 4-round zero-knowledge proofs using standard techniques (using a commitment scheme). Henceforth, it is not uncommon to use Σ-protocol to describe zero-knowledge proofs as its conceptually simpler. We are however interested in the so-called Fiat-Shamir transform [17].

Definition 4. Given a Σ-protocol (P, V) for a language $\mathcal{L}$ and a hash function $\mathcal{H}$ modeled as a random oracle, the *Fiat-Shamir transform* acts as follows:

1. The prover starts by providing a commitment a sampled honestly.
2. Then, the output of the verifier is computed using x and a as $c = \mathcal{H}(x, a)$.
3. Finally, the prover is given c and outputs an answer z.

The output of this transform is a proof $\pi = (a, z)$ that is verified by computing $c = \mathcal{H}(x, a)$ and then check that (a, c, z) is indeed an accepting transcript for the Σ-protocol (P, V).

Informally, this method enforces the challenge to be honestly generated. The zero-knowledge property is achieved using the simulator for the HVZK of the Σ-protocol using the fact that $\mathcal{H}$ is a random-oracle to argue that (a, c, z) is correctly distributed.

The tricky part is to prove the (computational) soundness, which is possible using the following lemma:

Lemma 1 (Forking lemma [26])**.** *Let Q_i denotes the random oracle queries. For $\frac{\varepsilon}{2T}$ fraction of $(q_1, \ldots, q_{i^\star})$ it holds that $\widehat{\mathsf{P}}$ wins in the soundness game with probability $\frac{\varepsilon}{2T}$ conditioned on $Q_{i^\star} = (x, a)$ and $Q_i = q_i$ for all $i \leq i^\star$.*

Which is admitted in the context of this course. Let us now prove the computational soundness of the Fiat-Shamir heuristic assuming the special soundness property of the Σ-protocol.

Proof. Let us assume that there is a PPT adversary $\widehat{\mathsf{P}}$ against the soundness game of the Fiat-Shamir transform that wins with probability ε in time T.

As this adversary is polynomial, there are a polynomial number of queries to the random oracle that are made: $(q_1, \ldots, qh)$. As $\mathcal{H}$ is modeled as a random oracle, we can then say that there is one of those queries that is a query on (x, a). Otherwise, it means that the adversary doesn't have any information on $\mathcal{H}(x, a)$ and cannot win with with probability greater than $1/|\mathcal{C}|$ ($\mathcal{C}$ denotes the challenge space, which is chosen so that $1/|\mathcal{C}|$ is negligible).

Thus, we can construct the following adversary against the soundness of the Σ-protocol: let us first run $\widehat{\mathsf{P}}$, which will eventually output a winning proof $(a^\star, z^\star)$ such that $(a^\star, \mathcal{H}(a^\star, x), z^\star)$ is a valid transcript for the underlying Σ-protocol. There exists an index $i^\star$ such that $\mathcal{H}(a^\star, x)$

is queried. Let us now rewind $\widehat{\mathsf{P}}$ with the same random coins and random oracle answers until the $i^\star$-th query, where we sample a new response $\mathcal{H}(a, x^\star) = \tilde{c}$ uniformly at random. Then by the Forking Lemma (Lemma 1), this new run succeeds with probability $\left(\frac{\varepsilon}{2T}\right)^2$ and it allows us to obtain a second accepting transcript $(a^\star, \tilde{c}, \tilde{z})$ that breaks the special soundness. □

2.3 Generic Construction of Group Signatures

In the following, we will describe the generic construction of zero-knowledge proofs proposed by Bellare, Micciancio and Warinschi [2]. Provided the following building blocks:

- a public-key encryption scheme PKE (its security will be defined later);
- an existentially unforgeable under chosen message attack ($\mathsf{EU\text{-}CMA}$) signature scheme Sig;
- a simulation-sound NIZK proof Π.

Let us define the following group signature:

Setup(λ, N)**:** Given the security parameter λ and the group size N, this algorithm does the following:

1. Sample $\mathsf{crs} \hookleftarrow \mathcal{U}(\{0,1\}^{p(\lambda)})$ for a polynomial p.
2. Generate keys $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{PKE.Keygen}(1^\lambda)$.
3. Generate keys $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Sig.Keygen}(1^\lambda)$.
4. For i from 1 to N:
   (a) Generate $(\mathsf{vk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Sig.Keygen}(1^\lambda)$.
   (b) Compute $\mathsf{cert}_i \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}, \langle \mathsf{vk}_i, i \rangle)$.
   (c) Set group secret key for user i as $\mathsf{gsk}[i] \leftarrow (i, \mathsf{vk}_i, \mathsf{sk}_i, \mathsf{cert}_i)$.
5. Define the group public key as $\mathsf{gpk} = (\mathsf{crs}, \mathsf{ek}, \mathsf{vk})$.
6. Set the opening key $\mathsf{ok} = (\mathsf{dk}, \mathsf{vk})$.

Sign$(\mathsf{gsk}[i], m)$**:** Given a user secret key $\mathsf{gsk}[i]$ parsed as $(i, \mathsf{vk}_i, \mathsf{sk}_i, \mathsf{cert}_i)$ and a message m:

1. Compute the signature $s \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}_i, m)$.
2. Compute the ciphertext $c \leftarrow \mathsf{PKE.Enc}(\mathsf{ek}, \langle i, \mathsf{vk}_i, \mathsf{cert}_i, s \rangle; r)$ with explicit randomness $r \hookleftarrow \mathcal{U}(\{0,1\}^{q(\lambda)})$.
3. Generate a proof π for the following statement:
   - c is an encryption of $\langle i, \mathsf{vk}_i, \mathsf{cert}_i, s \rangle$ under randomness r such that:
     – $\mathsf{Sig.Verify}(\mathsf{vk}, \langle i, \mathsf{vk}_i \rangle, \mathsf{cert}_i) = 1$.
     – $\mathsf{Sig.Verify}(\mathsf{vk}_i, m, s) = 1$.

   The corresponding language $\{(\langle \mathsf{ek}, \mathsf{vk}, m, c \rangle, \langle i, \mathsf{vk}_i, \mathsf{cert}_i, r \rangle)\}$ is denoted $\mathcal{L}_{\mathsf{gs}}$.
4. Output $\sigma = (c, \pi)$

Verify$(\mathsf{gpk}, m, \sigma)$**:** To verify a signature $\sigma = (c, \pi)$ on message m, this algorithm accepts if and only if $\Pi.\mathsf{Verify}(\langle \mathsf{ek}, \mathsf{vk}, m, c \rangle, \pi)$ accepts.

Open(ok, m, σ)**:** To open a signature $\sigma = (c, \pi)$ on message m with the opening key $\mathsf{ok} = (\mathsf{dk}, \mathsf{vk})$, the opening algorithm does the following if $\mathsf{Verify}(\mathsf{gpk}, m, \sigma)$ returns accept:

1. Parse $\mathsf{PKE.Dec}(\mathsf{dk}, c)$ as $\langle i, \mathsf{vk}_i, \mathsf{cert}_i, s\rangle$.
2. Return the identity i.

The correctness of the above scheme immediately follows from the completeness of the proof system. Let us now prove the different security properties of this group signature scheme.

Theorem 1. *If* Sig *is an EU-CMA signature scheme, and the NIZK* Π *is sound, then the above construction is* traceable.

Proof. Let $\mathcal{B}$ be an adversary that wins the $\text{Experiment}_{\text{traceability}}$ with non negligible probability with $\mathcal{Q}_u$ be the set of corrupted users as per Figure 1.

Let us define three types of forgeries:

- Type 0: The forgery $(m^\star, (c^\star, \pi^\star))$ is such that $\langle \mathsf{ek}, \mathsf{vk}, m^\star, c^\star\rangle \notin \mathcal{L}_{\mathsf{gs}}$.
- Type I: $\langle \mathsf{vk}^\star, i^\star\rangle$ has not been certified during the setup phase.
- Type II: $\langle \mathsf{vk}^\star, i^\star\rangle$ has been certified but $i \notin \mathcal{Q}_u$ and $(m^\star, i^\star)$ has not been queried to the signing oracle.

If the forgery is of type 0, then $\pi^\star$ has to prove a false statement. Otherwise, $c^\star$ would encrypt a message of the form $\langle i^\star, \mathsf{vk}_i^\star, \mathsf{cert}_i^\star, s\rangle$ where $\mathsf{Sig.Verify}(\mathsf{vk}, \langle i^\star, \mathsf{vk}_i^\star\rangle, \mathsf{cert}_i^\star) = 1$ and $\mathsf{Sig.Verify}(\mathsf{vk}_i^\star, m^\star, s^\star) = 1$, which means that $\langle \mathsf{ek}, \mathsf{vk}, m^\star, c^\star\rangle \in \mathcal{L}_{\mathsf{gs}}$ which contradicts our assumption. Therefore, if a Type 0 forgery happens, then it contradicts the soundness of the NIZK Π. Hence:

$$\Pr[\text{Type 0}] \leq 2^{-\lambda}.$$

If the forgery $(m^\star, (c^\star, \pi^\star))$ is a valid Type I forgery, then we can use the following adversary against the EU-CMA security of the signature scheme. The signing operations using sk are forwarded to the challenger against EU-CMA security. Using the opening key ok to decrypt $c^\star$, it is possible to obtain $\langle i^\star, \mathsf{vk}^\star, \mathsf{cert}^\star, s^\star\rangle$. Hence, $(\langle vk^\star, i^\star\rangle, \mathsf{cert}^\star)$ is a valid forgery against the EU-CMA security of the signature scheme Sig as it has never been signed under sk. Thus it holds that

$$\Pr[\text{Type I}] = \mathsf{Adv}^{\text{eu-cma}}_{\mathcal{A}_1}(\lambda).$$

Finally, if the forgery is of Type II, then it is possible to produce the following algorithm against EU-CMA. First guess the identity $i' \in [N]$ uniformly at random. Then all the keys $\mathsf{gsk}[j]$ are generated honestly for $j \neq i'$. Then, upon interacting with $\mathcal{B}$, the reduction does the following.

- If $\mathcal{B}$ requests $\mathsf{gsk}[i]$, then abort.
- On oracle queries (j, m) with $j \neq i'$, it is possible to compute $s = \mathsf{Sign}(\mathsf{gsk}[j], m)$
- On oracle queries (i', m): call signing oracle on m.

Then, upon receiving the forgery $(m^\star, (c^\star, \pi^\star))$, the reduction first decrypts $c^\star$ into $\langle i^\star, \mathsf{vk}^\star, \mathsf{cert}^\star, s^\star\rangle$ and outputs $m^\star, s^\star$ as a forgery.

This reduction successes with probability $1/N$ (the guessing probability), meaning that:

$$\frac{1}{N}\Pr[\text{Type II}] = \mathsf{Adv}^{\text{eu-cma}}_{\mathcal{A}_2}(\lambda).$$

Hence, to sum up, the advantage of any PPT against the tracing algorithm is bounded by:

$$\mathsf{Adv}_{\mathcal{B}}^{\text{traceability}}(\lambda) \leq 2^{-\lambda} + \mathsf{Adv}_{\mathcal{A}_1}^{\text{eu-cma}}(n) + N \cdot \mathsf{Adv}_{\mathcal{A}_2}^{\text{eu-cma}}(\lambda).$$

□

Theorem 2. *If* PKE *is an IND-CPA (resp. IND-CCA) secure public-key encryption scheme, and* Π *is a simulation-sound ZK proof system, then the above construction is* anonymous *(resp.* fully-anonymous*).*

In the following, we will provide the proof for full anonymity.

Proof. Let us assume we have an adversary $\mathcal{B}$ against the anon experiment and let us interact with it in order to break the security of the PKE encryption scheme. Let us first make a modification on the experiment $\text{Experiment}_{\text{anon}}$. Instead of generating crs uniformly at random, it is generated by the ZK simulator S_1 along with its secret trapdoor τ_{crs}. Then on step 3 on the signature scheme, this trapdoor τ_{crs} is used to generate simulated proofs π to answer signature queries. This does not change the view of the adversary by ZK properties, then the difference with the initial game is negligible.

We then use this modified scheme to construct a reduction to the security of the CCA game. Let us assume that we are interacting with a challenger for the CCA game, it first starts by providing the reduction an encryption key ek. This key is embedded as is in the group secret key in the setup phase of the group signature, the rest is generated as in our modified scheme.

Signature queries. Upon receiving signature queries, the reduction uses the secret key $\mathsf{gsk}[i]$ to faithfully answer the request (with the proof generated with trapdoor τ_{crs}.

Opening queries. When the reduction receives an opening query about signature $\sigma = (c, \pi)$ about message m, it sends c to the decryption oracle of the CCA challenger and upon receiving the answer parsed as $\langle i, \mathsf{vk}, \mathsf{cert}, _s \rangle$, it sends back the identity i to the adversary $\mathcal{B}$.

Challenge phase. Then, during the challenge phase, the adversary $\mathcal{B}$ sends a challenge m, i_0, i_1 to our reduction. The reduction then prepares two messages for $b \in \{0, 1\}$: $M_b = \langle i_b, \mathsf{vk}_b, \mathsf{cert}_b, \mathsf{Sig.Sign}(\mathsf{sk}_{i_b}, m)\rangle$ and sends M_0 and M_1 to the CCA adversary as a challenge query. The reduction then receives a challenge $c^\star$, for which it computes a simulated proof $\pi^\star \leftarrow \Pi.\mathsf{S}_2(\tau_{\mathsf{crs}}, \langle \mathsf{ek}, \mathsf{vk}, m, c^\star)$ before sending $(c, \pi^\star)$ as a challenge signature to the adversary $\mathcal{B}$. $\mathcal{B}$ will then do other computations, and if it asks for a valid opening query for $(c^\star, \pi')$, then the reduction aborts. Otherwise, $\mathcal{B}$ will eventually return an answer d to its challenge which is forwarded ot the CCA adversary.

Analysis. We first note that the reduction aborts with probability $\mathsf{Adv}_{\mathcal{A}}^{\text{sim-sound}}(\lambda)$, as it means that the adversary broke the soundness under the view of simulated transcripts, and thus the simulation soundness. Which means that the advantage of our reduction can be written down as:

$$\mathsf{Adv}_{\mathcal{C}}^{\text{cca}}(\lambda) = \left|\left(\Pr[d = 1 \mid b = 1] - \Pr[d = 1 \mid b = 0]\right)(1 - \Pr[\text{abort}])\right|,$$

meaning that, after taking into consideration our initial change that makes the resulting game differing from the original one by $\mathsf{Adv}_{\mathcal{A}'}^{\text{zk}}(\lambda)$:

$$\mathsf{Adv}_{\mathcal{B}}^{\text{anon}}(\lambda) = \frac{\mathsf{Adv}_{\mathcal{C}}^{\text{cca}}(\lambda)}{1 - \mathsf{Adv}_{\mathcal{A}}^{\text{sim-sound}}(\lambda)} + \mathsf{Adv}_{\mathcal{A}'}^{\text{zk}}(\lambda).$$

□

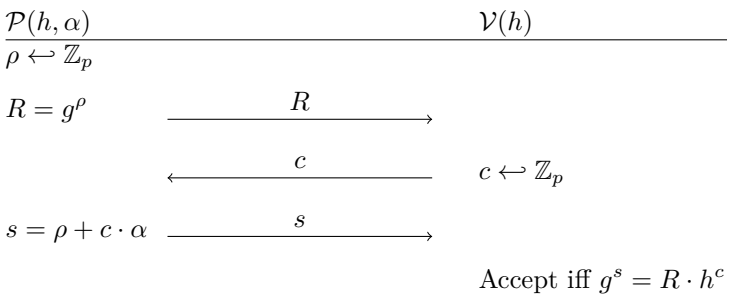

Figure 3: Schnorr protocol for discrete logarithm

2.4 Toward Instantiations of Group Signatures

As we saw, the key component of this construction is the proof at step 3 of the signing algorithm. This is a proof of correct encryption, and knowledge of valid signatures on the message under this ciphertext. In this section we will address the first part, and explain why the second part requires more works.

First of all, let us consider the following encryption scheme, the El Gamal encryption [16]. This encryption scheme, that relies on the decisional Diffie-Hellman assumption, uses as public parameters pp a cyclic group $\mathbb{G}$ of prime order p along with a generator g.

Setup$(1^\lambda, \mathsf{pp})$**:** The setup algorithm starts by sampling a decryption key $\mathsf{dk} \triangleq \alpha \hookleftarrow \mathbb{Z}_p$ and computes $\mathsf{ek} \triangleq h = g^\alpha \in \mathbb{G}$.

Encrypt$(\mathsf{ek}, m; r)$**:** To encrypt a message $m \in [1, B_m]$ using encryption key $\mathsf{ek} = h$ and a randomness $r \in \mathbb{Z}_p$, this algorithm computes the following two parts:

$$c_1 = g^m \cdot h^r, \qquad c_2 = g^r.$$

And sends (c_1, c_2) as the ciphertext.

Decrypt(dk, c)**:** To decrypt a ciphertext of the form $c = (c_1, c_2)$ using the decryption key $\mathsf{dk} = \alpha$, this algorithm first computes:

$$g^{m'} = c_1 \cdot c_2^{-\alpha},$$

given that B_m is polynomially large, a discrete-log algorithm (let's say giant-step baby-step or Pollard-ρ) can be used to recover m' that is thus returned.

Then, we give a proof of correct decryption using a Schnorr-like protocol [28]. Let us recall that the Schnorr protocol for discrete logarithm is a three-round protocol to prove the knowledge of an exponent α such that g^α is equal to some public value h. The protocol for discrete log is described in Figure 3. In our case, we use it with the Fiat-Shamir heuristic to make the proof non-interactive (see Definition 4).

The resulting construction to prove the knowledge of a message m under a ciphertext (c_1, c_2) and public key h is described as follows, using a hash function $\mathcal{H}$ that is modeled as a random oracle:

1. First sample randomnesses ρ_m, ρ_r from $\mathbb{Z}_p$ and compute

$$R_1 = g_m^\rho h_r^\rho, \qquad R_2 = g_r^\rho,$$

2. Compute $c = \mathcal{H}(R_1, R_2, c_1, c_2, h)$, then

$$s_r = \rho_r + c \cdot r, \qquad s_m = \rho_m + c \cdot m.$$

3. Send (R_1, R_2, s_r, s_m) as the proof π.

To verify such a proof π, the verifier checks that the following equations hold:

$$R_1 = g^{s_m} \cdot h^{s_r} c_1^{-c}, \qquad R_2 = g^{s_r} \cdot c_2^{-c}, \qquad c = \mathcal{H}(R_1, R_2, c_1, c_2, h).$$

However we are still missing a key component: zero-knowledge proofs compatible with signatures. Our tentative construction ends here as there is no construct from standard DDH-like assumptions. However it is possible to obtain them from pairings for instance [23, 24, 22], strong-RSA [8], or lattice assumptions [21]. Let us notice that these signature schemes are not allowed to hash the message as it breaks its structure and makes it unusable for zero-knowledge proofs. They rely on commitment schemes to be able to be able to compute on hidden data.

Informally, a commitment scheme is the digital equivalent of a safe box: one can commit a value, and once it's inside a safe, it is impossible to modify what is inside the box (binding property) nor know what's inside (hiding property). More formally, it is defined as follows.

Definition 5 (Commitment Scheme)**.** A *commitment scheme* is a triple of algorithms $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$ such that:

Setup(1^λ)**:** This algorithm takes as input a security parameter λ and outputs public parameters pp.

Commit(pp, m)**:** On inputs public parameters pp and a message m, this algorithm outputs a commitment com and a decommitment dec

Open$(\mathsf{pp}, m, \mathsf{com}, \mathsf{dec})$**:** Given public parameters pp, a message m, a commitment com and a decommitment dec this algorithm returns accept or reject.

Moreover, a commitment is required to verify the following security properties:

Binding. A commitment scheme is *computationally binding* if any PPT algorithm $\mathcal{A}$ can win the following game with non-negligible probability:

1. First the challenger runs the Setup algorithm and sends pp to the adversary.
2. The adversary $\mathcal{A}$ outputs a commitment com and two openings (m, dec), (m', dec') and wins if

$$\mathsf{Open}(\mathsf{pp}, m, \mathsf{com}, \mathsf{dec}) = \mathsf{accept}, \quad \mathsf{Open}(\mathsf{pp}, m', \mathsf{com}, \mathsf{dec}') = \mathsf{accept} \quad \text{and } m \neq m'.$$

If the adversary is unbounded, the scheme is said to be *perfectly binding*.

Hiding. A commitment scheme is *hiding* if the following distributions are indistinguishable for any m_0, m_1: $\{\mathsf{com} \mid (\mathsf{com}, \mathsf{dec}) \leftarrow \mathsf{Com}(\mathsf{pp}, m_0)\}$ and $\{\mathsf{com} \mid (\mathsf{com}, \mathsf{dec}) \leftarrow \mathsf{Com}(\mathsf{pp}, m_1)\}$. If they are statistically indistinguishable, the scheme is said to be *perfectly hiding*, and *computationally hiding* if they are computationally indistinguishable.

An example of commitment scheme is Pedersen's commitment [25]:

Setup(1^λ)**:** Choose a cyclic group $\mathbb{G}$ of prime order $p > 2^\lambda$ and $g, h \hookleftarrow \mathcal{U}(\mathbb{G})$ and output $\mathsf{pp} = (\mathbb{G}, g, h)$.

Commit(pp, m)**:** Sample a random $r \hookleftarrow \mathcal{U}(\mathbb{Z}_p)$ and output $\mathsf{com} := g^m \cdot h^r$ and $\mathsf{dec} := (m, r)$.

Open$(\mathsf{pp}, m, \mathsf{com}, \mathsf{dec})$**:** Parse dec as (m', r) and accept if and only if $\mathsf{com} = g^m h^r$ and $m = m'$.

Security. This scheme is perfectly hiding, as h^r is distributed uniformly in $\mathbb{G}$ and acts as a one-time-pad.
For the binding property, it relies on the difficulty of the discrete logarithm problem (DLP). Given a DLP instance $\mathbb{G}, g, h$, we send it as public parameters to the adversary $\mathcal{A}$ against the Pedersen commitment's binding, which is distributed correctly. The adversary $\mathcal{A}$ eventually outputs two distinct pairs (m, r) and (m', r') such that $g^m h^r = g^{m'} h^{r'}$, which means that $\log_g(h) = \frac{m'-m}{r-r'} \bmod p$, is the discrete logarithm we were asked to solve. We moreover notice that as $m \neq m' \bmod p$, then $r \neq r' \bmod p$, hence the inverse of $r - r'$ exists in $\mathbb{Z}_p$.

2.5 Variations of Group Signatures

Since their introduction, group signatures have been extended through different manners in order to extend either their functionality or the anonymity guarantees. We already hinted it via dynamically group signatures, to allow users to interactively enroll into the group at any point of time (without altering the group public key). Here, we will start with a simpler example of such extensions with group *signatures with message dependent opening* (GS-MDO) [27].

Recalling the public transportation example, we noticed that the opening authority possesses a lot of power, and if it is corrupted, then the anonymity property is meaningless. This flavour introduces a new authority: the admitter which issues tokens on specific messages. These tokens are necessary to lift anonymity on these messages. Hence, the accountability is ensured by the collaboration of the opening authority and the admitter, while neither of them can reveal the identity of a signer alone. In this context, in case of an incident, the admitter can issue a token for the timestamps related to the incident.

Definition 6 (GS-MDO)**.** A *group signature with message-dependent opening* is a tuple of 5 algorithms $(\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Trapgen}, \mathsf{Open})$ acting as follows:

Setup$(1^\lambda, N)$**:** this algorithm takes as inputs the security parameter λ and the number of users N and outputs the group public key gpk, secret keys $(\mathsf{gsk}[i])_{i=1}^N$ and the opening key ok and the admitter secret key $\mathsf{mks}_{\mathrm{adm}}$.

Sign$(\mathsf{gsk}[i], m)$**:** it takes as inputs a user secret key $\mathsf{gsk}[i]$, a message m to be signed and outputs a signature σ.

Verify$(\mathsf{gpk}, m, \sigma)$**:** given a message m, a signature σ and the group public key gpk, this algorithm returns true or false.

Trapgen$(\mathsf{msk}_{\mathrm{adm}}, m)$**:** with the admitter secret key $\mathsf{msk}_{\mathrm{adm}}$ and a message, this algorithm outputs a token τ_m.

Open$(\mathsf{ok}, m, \tau_m \sigma)$**:** given a message m, a token τ_m, a signature σ and the group public key gpk, this algorithm returns true or false.

In the security requirements, the notion of anonymity is separated in two different notion: the anonymity against the admitter and the anonymity against the opening authority. These notions are described in the security games where only one of the two secret keys is given to the adversary according to their name. The traceability remains unchanged (adapting the opening queries to integrate $\mathsf{Trapgen}$).

A natural way to construct them is using two different layers of encryption around the identity in the signature algorithm: the outer one for the admitter and the inner one for the opening authority. This is summarised in Figure 4. However, we notice that by doing that, it still not depend on the message. To do this, we use an identity-based encryption (IBE) scheme. An IBE scheme is an encryption scheme, with an extra authority that possesses a master secret-key that is used to derive secret keys for users under some identity id (for instance an e-mail address).

Definition 7. An IBE scheme is a tuple of algorithms $(\mathsf{Setup}, \mathsf{Keygen}, \mathsf{Enc}, \mathsf{Dec})$ such that:

$\mathsf{Setup}(1^\lambda)$: from the security parameter λ, this algorithm outputs public parameters pp and a master secret key msk.

$\mathsf{Keygen}(\mathsf{msk}, \mathsf{id})$: given the master secret key msk and an identity string id, this algorithm outputs the user secret key $\mathsf{sk}_{\mathsf{id}}$.

$\mathsf{Enc}(\mathsf{pp}, \mathsf{id}, M)$: using the public parameters pp, an identity id and a message M, this algorithm provides a ciphertext C.

$\mathsf{Dec}(\mathsf{sk}_{\mathsf{id}}, C)$: the ciphertext C is decrypted using $\mathsf{sk}_{\mathsf{id}}$ by this algorithm that outputs a message M.

While its correctness is straightforward, an IBE have many different possible security requirements. The security can be active or passive with respect of the decryption oracle, similarly to a PKE. However, the adversary have also access to a key derivation oracle, which can be either selective (the corrupted identities are known at the outset of the security game), or adaptive, where the queries are made adaptively. While the former security requirement is unrealistic, it leads to more efficient constructions which are sufficient to be used as a building block.

To encrypt a message, one only needs to know the identity of the receiver (and the public parameters), while decrypting requires the secret key associated to the identity under which the message has been encrypted. Initially introduced to avoid public-key infrastructures, identity-based encryption is a very versatile building block in cryptography, which can be used to design signatures or encryption keys.

To design an GS-MDO scheme, we use an IBE as the outer encryption, and the signer encrypts its identity with an IBE, where the identity of the IBE scheme is the message to be signed. Hence, the secret-key for this message can be issued by the admitter (which possesses the master secret key for the IBE scheme).

Identity-Based Encryption. An example of IBE scheme based on pairings is the Boneh-Franklin IBE [5] that relies on pairings.

Definition 8. Let G, G_T two cyclic groups of prime order $p > 2^\lambda$ and g a generator of G. A (symmetric) pairing is a map $e : G \times G \to G_T$ that verifies:

- **Bilinearity**: for any $a, b \in \mathbb{Z}_p$, it holds that $e(g^a, g^b) = e(g, g)^{ab}$;

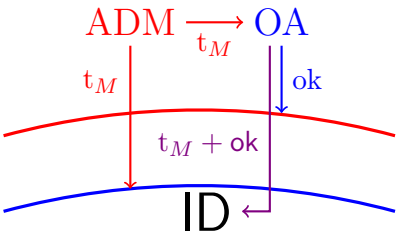

Figure 4: Two-layer encryption of the identity for GS-MDO.

- **Non-degeneracy**: $e(g^a, g^b) = 1 \implies a = 0 \vee b = 0$;
- **Efficiently computable**.

The security of the Boneh-Franklin IBE relies on the following security assumption, which is akin to DDH in the G_T.

Definition 9. The *decision bilinear Diffie-Hellman* (DBDH) problem is given a pairing $e : G \times G \to G_T$ and a generator g of G, given (g, g^a, g^b, g^c, T), decide if $T = e(g, g)^{abc}$ or if $T \in_R G_T$.

The DBDH assumption is that no PPT algorithm can solve the DBDH problem with non-negligible advantage.

Setup(1^λ)**:** from the security parameter λ, pick a bilinear map $e : G \times G \to G_T$ of prime order $p \geq 2^\lambda$ and a generator $g \hookleftarrow G$. Pick a hash function $H : \{0,1\}^\star \to G$ and finally pick a random $\alpha \hookleftarrow \mathbb{Z}_p$. The master secret key msk is defined as α and the public parameters are $\mathsf{pp} := \{e, (G, G_T), g, h = g^\alpha, H\}$.

Keygen$(\mathsf{msk}, \mathsf{id})$**:** the secret key for user id is computed as $H(\mathsf{id})^\alpha$.

Enc$(\mathsf{pp}, \mathsf{id}, M)$**:** to encrypt a message for identity id, pick a random $r \hookleftarrow \mathbb{Z}_p$ and output

$$(c_1, c_2) := (g^r, M \cdot e(h, H(\mathsf{id}))^r).$$

Dec$(\mathsf{sk}_{\mathsf{id}}, C)$**:** to decrypt a ciphertext (c_1, c_2) given $H(\mathsf{id})^\alpha)$, output $c_2 \cdot H(c_1, H(\mathsf{id})^\alpha)^{-1}$.

Theorem 3. *The Boneh-Franklin IBE is IND-ID-CPA secure in the random oracle model under the decision bilinear Diffie-Hellman assumption.*

Proof. Let $\mathcal{A}$ be an adversary against the Boneh-Franklin IBE that wins the IND-ID-CPA game with non-negligible advantage ε.

Let us build a reduction $\mathcal{B}$ that wins the DBDH game with non-negligible probability. At first, the reduction receives a DBDH challenge (g, g^a, g^b, g^c, T) and has to devise either $T = e(g, g)^{a,b,c}$ or is a random element in G_T. To do this, the reduction $\mathcal{B}$ sends to $\mathcal{A}$ the public parameters $\mathsf{pp} := (e, (G, G_T), g, h := g^a, H)$ (implicitly setting $\alpha = a$). Then $\mathcal{B}$ interacts with $\mathcal{A}$ as follows.

Hash queries. When $\mathcal{A}$ queries the value $H(x)$, it is answered as follows:

- Either x have been queried before and it returns the same value $H(x)$.

- Otherwise, $\mathcal{B}$ sample a bit b_x such that $b_x = 1$ with probability $1/q + 1$ where q is (an upper bound on) the number of private-key queries, and a random value $\beta_x \hookleftarrow \mathbb{Z}_p$.
  - If $b_x = 0$, define $H(x) = g^{\beta_x}$.
  - If $b_x = 1$, define $H(x) = (g^b)^{\beta_x}$.

  In both cases, store (x, b_x, β_x) in a list $\mathcal{Q}_h$ initially empty.

Private-key queries. When $\mathcal{A}$ queries private-key $\mathsf{sk}_{\mathsf{id}}$ for identity id, we first assume w.l.o.g. that a hash query is made before asking the private-key query. Hence $\mathcal{B}$ can recover the entry $(\mathsf{id}, b_{\mathsf{id}}, \beta_{\mathsf{id}})$ from $\mathcal{Q}_h$.

- If $b_{\mathsf{id}} = 1$, abort and outputs a random bit d.
- If $b_{\mathsf{id}} = 0$, $\mathcal{B}$ can compute $H(\mathsf{id})^{\alpha} = (g^{\alpha})^{\beta_{\mathsf{id}}}$.

Challenge query. At some point $\mathcal{A}$ issues a challenge query $(M_0, M_1, \mathsf{id}^{\star})$ where $\mathsf{id}^{\star}$ has not been queried before as a private-key query. We assume w.l.o.g. that $H(\mathsf{id}^{\star})$ has been queried before as $\mathcal{B}$ can make the query itself otherwise. Therefore, $\mathcal{B}$ can recover $(\mathsf{id}^{\star}, b_{\mathsf{id}^{\star}}, \beta_{\mathsf{id}^{\star}})$ from $\mathcal{Q}_h$.

- If $b_{\mathsf{id}^{\star}} = 0$, abort and output a random bit d.
- Otherwise, $\mathcal{B}$ picks a random bit γ and set the challenge ciphertext as:

$$c^{\star} = (g^c, M_{\gamma} \cdot T^{\beta_{\mathsf{id}^{\star}}})$$

We note that in either case, $c^{\star}$ is distributed uniformly at random in $G \times G_T$ from the point of view of $\mathcal{A}$, which is the distribution of an encryption.

Output. Eventually, $\mathcal{A}$ outputs a bit γ'. If $\gamma = \gamma'$, then $\mathcal{B}$ outputs 1 (meaning that $T = e(g, g)^{abc}$), otherwise it outputs 0 (meaning that $T \in_R G_T$).

Analysis. Firstly, let us evaluate the probability that $\mathcal{B}$ doesn't abort, which is:

$$\begin{aligned}
\Pr[\neg\mathsf{abort}] &= \Pr[b_{\mathsf{id}^{\star}} = 1] \cdot \Pr\left[\bigwedge_{i=1}^{q} b_{x_{\mathsf{id}_i}} = 0\right] \\
&= \frac{1}{q+1} \cdot \left(1 - \frac{1}{q+1}\right)^q \\
&\approx \frac{1}{e^1 \cdot (q+1)},
\end{aligned}$$

for large enough q.

Now,

$$\begin{aligned}
\Pr[\mathcal{B}=1 \mid T=e(g,g)^{abc}] &= \frac{\Pr[\mathcal{B}=1 \wedge T=e(g,g)^{abc} \wedge \neg\mathsf{abort}]}{\Pr[T=e(g,g)^{abc}]} \cdot \frac{Pr[\neg\mathsf{abort} \wedge T=e(g,g)^{abc}]}{Pr[\neg\mathsf{abort} \wedge T=e(g,g)^{abc}]} \\
&+ \frac{\Pr[\mathcal{B}=1 \wedge T=e(g,g)^{abc} \wedge \mathsf{abort}]}{\Pr[T=e(g,g)^{abc}]} \cdot \frac{Pr[\mathsf{abort} \wedge T=e(g,g)^{abc}]}{Pr[\mathsf{abort} \wedge T=e(g,g)^{abc}]} \\
&= \Pr[\mathcal{B}=1 \mid T=e(g,g)^{abc} \wedge \neg\mathsf{abort}] \cdot \Pr[\neg\mathsf{abort} \wedge T=e(g,g)^{abc}] \\
&+ \Pr[\mathcal{B}=1 \mid T=e(g,g)^{abc} \wedge \mathsf{abort}] \cdot \Pr[\mathsf{abort} \wedge T=e(g,g)^{abc}] \\
&= \Pr[\mathcal{B}=1 \mid T=e(g,g)^{abc} \wedge \neg\mathsf{abort}] \cdot \Pr[\neg\mathsf{abort} \wedge T=e(g,g)^{abc}] \\
&+ \frac{1}{2} \cdot \Pr[\mathsf{abort}] \\
&= \frac{1}{2} + \Pr[\neg\mathsf{abort}] \cdot \left(\Pr[\mathcal{B}=1 \mid T=e(g,g)^{abc} \wedge \neg\mathsf{abort}] - \frac{1}{2}\right).
\end{aligned}$$

Similarily,

$$\Pr[\mathcal{B}=1 \mid T \in_R G_T] = \frac{1}{2} + \Pr[\neg\mathsf{abort}] \left(\Pr[B=1 \mid T \in_R G_T \wedge \neg\mathsf{abort}] - \frac{1}{2}\right) = \frac{1}{2}$$

Thus, putting everything together gives us the probability that the reduction wins as:

$$\begin{aligned}
\mathsf{Adv}_{\mathcal{B}}^{\mathrm{dbdh}}\lambda &= \left|\Pr[\mathcal{B}=1 \mid T=e(g,g)^{abc}] - \Pr[\mathcal{B}=1 \mid T \in_R G_T]\right| \\
&= \varepsilon \Pr[\neg\mathsf{abort}] \\
&= \frac{\varepsilon}{e^1(q+1)}.
\end{aligned}$$

□

The modifications with the previous construction are as follows:

- In the key generation algorithm, a key pair $(\mathsf{msk}, \mathsf{mpk})$ are generated for an IBE scheme and the mpk is included in the group public key, while the master secret key is the admitter secret key.
- During the signature phase, the identity of the user is first signed under the opening authority key, then this encryption is encrypted using the IBE under the identity m.
- Finally, the $\mathsf{Trapgen}$ algorithm outputs the IBE's secret key for the message m as the token.

2.6 Dynamic Group Signatures

To finish this part about group signatures, we will present dynamic group signatures in this section (that has already been hinted previously). Two equivalent models have been proposed by Kiayias and Yung [20] and Bellare, Shi and Zhang [3].

Definition 10 (Group Signatures)**.** A dynamic group signature is a tuple of algorithms or protocols $(\mathsf{Setup}, \mathsf{Join}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Open})$ acting as follows:

Setup$(1^\lambda, N)$**:** this algorithm takes as inputs the security parameter λ and an upper bound on the number of users N and outputs the group public key gpk, the group master secret key msk and the opening key ok.

Join: this interactive protocol is run between an enrolling user knowing its identity and the group manager possessing the group master secret key. At the end of the interactions, the user obtains its secret key $\mathsf{gsk}_{\mathsf{id}}$ and the transcript of the interaction is publicly available.

Sign$(\mathsf{gsk}_{\mathsf{id}}, m)$**:** it takes as inputs a user secret key gskid, a message m to be signed and outputs a signature σ.

Verify$(\mathsf{gpk}, m, \sigma)$**:** given a message m, a signature σ and the group public key gpk, this algorithm returns true or false.

Open(ok, m, σ)**:** given a message m, a signature σ and the group public key gpk, this algorithm returns true or false.

In the security definitions, another security requirements is necessary: non-frameability. Which summarized the fact that either knowing the group secret key, no one is able to frame an honest user by forging a signature that opens to it.

3 Electronic-Cash

Introduced by Chaum in 1982, e-cash initially aims at providing the digital equivalent of real-world currencies with a focus on users' privacy. The bank issues coins that can be withdrawn by users and spent to merchants. Eventually, the money is latter deposit to accounts within the bank. Between the moment a coin is withdrawn and deposited again, it should be impossible to identify how it has been spent, contrary to other electronic payment systems.

Since its inception, many variants have been proposed, featuring different properties, such as divisible e-cash, that allows splitting a coin (while standard e-cash possesses the same drawback as traditional coin: that is, the merchant has to possess the coins to provide exact change), decentralised versions and so on.

3.1 Definition

In this e-cash scenario, there are three players: the user, the bank and the merchant.

Definition 11 (Compact e-cash [6])**.** An e-cash system is a tuple of algorithms or protocols (BKeygen, UKeygen, Withdraw, Spend, Deposit, Identify, VerifyGuilt, Trace, VerifyOwnership) such that:

BKeygen$(1^\lambda, params)$**:** it is the key generation algorithm for the bank $\mathcal{B}$. It takes as input the security parameter λ, and some common parameters $params$. This algorithm outputs a key pair $(\mathsf{pk}_{\mathcal{B}}, \mathsf{sk}_{\mathcal{B}})$.

UKeygen$(1^\lambda, params)$**:** it is the key generation algorithm for the user $\mathcal{U}$ that outputs $(\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})$. Since merchants are a subset of users, they may use the same algorithm as well to obtain their keys.

Withdraw$(\mathcal{U}(\mathsf{pk}_{\mathcal{B}}, \mathsf{sk}_{\mathcal{U}}, n), \mathcal{B}(\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{B}}, n))$**:** this protocol between the bank $\mathcal{B}$ and the user $\mathcal{U}$ allows the latter to withdraw a wallet W of n coins from the bank $\mathcal{B}$. At the end of the interaction, the user outputs a wallet W or an error message $\perp$ while the bank outputs some message $\mathcal{T}_W$ that allows it to trace the user should this user double-spend some coin or an error message $\perp$. The bank maintains a database D for this trace information, to which it enters the record $(\mathsf{pk}_{\mathcal{U}}, T_W)$.

Spend$(\mathcal{U}(W, \mathsf{pk}_{\mathcal{M}}), \mathcal{M}(\mathsf{sk}_{\mathcal{M}}, \mathsf{pk}_{\mathcal{B}}, n))$**:** in this protocol, a user $\mathcal{U}$ gives one coin of its wallet W to the merchant $\mathcal{M}$. Here, the merchant obtains a serial number S of the coin and a proof π of validity of the coin. The user's output is an updated wallet W'.

Deposit$(\mathcal{M}(\mathsf{sk}_{\mathcal{M}}, S, \pi, \mathsf{pk}_{\mathcal{B}}), \mathcal{B}(\mathsf{sk}_{\mathcal{M}}, \mathsf{sk}_{\mathcal{B}}))$**:** in this protocol, a merchant $\mathcal{M}$ deposits a coin (S, π) into its account held by the bank $\mathcal{B}$. Whenever an honest $\mathcal{M}$ obtained (S, π) by running the Spend protocol with any (honest or otherwise) user, there is a guarantee that this coin will be accepted by the bank $\mathcal{B}$. Hereafter, $\mathcal{B}$ adds (S, π) to the list of spent coins. The merchant returns either nothing or an error message $\perp$.

Identify$(params, S, \pi_1, \pi_2)$**:** given two proof of validity π_1, π_2 for a serial number S, this algorithm outputs a public key $\mathsf{pk}_{\mathcal{U}}$ and a proof Π_G. If the merchant $\mathcal{M}$ who submitted the two proofs is *not* malicious, then Π_G is a proof that $\mathsf{pk}_{\mathcal{U}}$ is the registered public key of the user that double-spent the coin S.

VerifyGuilt$(params, S, \mathsf{pk}_{\mathcal{U}}, \Pi_G)$**:** this algorithm allows to publicly verify proof Π_G that the user with public key $\mathsf{pk}_{\mathcal{U}}$ is guilty of double-spending coin S.

Trace$(params, S, \mathsf{pk}_{\mathcal{U}}, \Pi_G, D, n)$**:** given a public key $\mathsf{pk}_{\mathcal{U}}$ of a double-spender, a proof Π_G of his guilt in double-spending coin S, the database D, and a wallet size n, computes the serial number $S_1, \ldots, S_m$ of all the coins issued to $\mathcal{U}$ along with proofs $\Pi_1, \ldots, \Pi_m$ of $\mathsf{pk}_{\mathcal{U}}$'s ownership. If VerifyGuilt$(params, S, \mathsf{pk}_{\mathcal{U}}, \Pi_G)$ doesn't accept, this algorithm does nothing.

VerifyOwnership$(params, S, \Pi, \mathsf{pk}_{\mathcal{U}}, n)$**:** this algorithm allows to publicly verify the proof Π that a coin with serial number S belongs to a double-spender with public-key $\mathsf{pk}_{\mathcal{U}}$.

This scheme should also verify the following security properties, that are initially provided informally.

Correctness. If an honest user runs Withdraw with an honest bank, then neither will output an error message. If an honest user runs Spend with an honest merchant, then the merchant accepts the coin.

Balance. From the bank's point of view, what matters is that no collection of users and merchants can ever spend more coins than they withdrew. It is required that there is a knowledge extractor $\mathcal{E}$ that executes u Withdraw protocols with an adversarial user and extracts un serial numbers $S_1, \ldots, S_{un}$. It is asked that for every adversary, the probability that an honest bank will accept (S, π) as the result of the Deposit protocol, where $S \neq S_i \forall i \in [1; un]$ is negligible. If $S_1, \ldots, S_n$ is a set of serial numbers output by $\mathcal{E}$ when running Withdraw with public key pk_{U}, we say that coins $S_1, \ldots, S_n$ belong to $\mathcal{U}$ identified by $\mathsf{pk}_{\mathcal{U}}$.

Identification of Double-Spenders. Suppose $\mathcal{B}$ is honest, $\mathcal{M}_1, \mathcal{M}_2$ are honest merchants who ran the Spend protocol with the adversary such that $\mathcal{M}_1$ outputs (S, π_1) and $\mathcal{M}_2$ outputs

S, π_2). This property guarantees that, with high probability, $\mathsf{Identify}(params, S, \pi_1, \pi_2)$ outputs a key $\mathsf{pk}_{\mathcal{U}}$ and proof Π_G such that $\mathsf{VerifyGuilt}(params, S, \mathsf{pk}_{\mathcal{U}}, \Pi_G)$ accepts.

Tracing of Double-Spenders. Given a user $\mathcal{U}$ is shown guilty of double-spending coin S by a proof Π_G s.t. VerifyGuilt accepts, this property ensures that $\mathsf{Trace}(params, S, \mathsf{pk}_{\mathcal{U}}, \Pi_G, D, n)$ will output the serial numbers $S_1, \ldots, S_n$ of all coins that belongs to $\mathcal{U}$ with proofs of ownership $\Pi_1, \ldots, \Pi_n$ such that for all i, with high probability, $\mathsf{VerifyOwnership}(params, S_i, \Pi_i, \mathsf{pk}_{\mathcal{U}}, n)$ also accepts.

Anonymity of Users. From a privacy point of view, what matters is that the bank, even when cooperating with any collection of malicious users and merchants, cannot learn anything about a user's spending other than what is available from side information from the environment. To capture this property more formally, we introduce a simulator $\mathcal{S}$, that possesses some side information not normally available to players. For instance, in the common parameters model, $\mathcal{S}$ generates these parameters; in the ROM, $\mathcal{S}$ controls the random oracle, etc. We require that $\mathcal{S}$ can create simulated coins without access to any wallet, such that a simulated coin is indistinguishable from a valid one. More precisely, $\mathcal{S}$ executes the user's side of the Spend protocol without access to the user's secret or public key, or his wallet W.

Exculpability. Suppose that an adversary that participates any number of times in the Withdraw protocol with the honest user with public key $\mathsf{pk}_{\mathcal{U}}$, and subsequently to that, in any number of legal Spend protocols with the same user. I.e., if the user withdrew u wallet of n coins each, then this user can participate in at most un Spend protocols. The adversary then outputs a coin serial number S and a purported proof Π that the user with public key $\mathsf{pk}_{\mathcal{U}}$ is a double-spender and owns coins S. The *weak* exculpability property requires that, for all adversary, the probability $\mathsf{VerifyOwnership}(params, S, \mathsf{pk}_{\mathcal{U}}, \Pi, n)$ accepts is negligible. Furthermore, the adversary may continue to engage in the user $\mathcal{U}$ in Spend protocols even if it means $\mathcal{U}$ must double-spend some coins of its choosing (in which case, the state of its wallet is reset). The adversary then outputs (S, Π). The *strong* exculpability property postulates that, for all adversaries, when S is a coin serial number *not* belonging to $\mathcal{U}$, the weak exculpability holds, and when S is a coin serial number *not* double-spent by user $\mathcal{U}$ with public key $\mathsf{pk}_{\mathcal{U}}$, the probability that $\mathsf{VerifyGuilt}(params, S, \Pi, \mathsf{pk}_{\mathcal{U}}, n)$ accepts is negligible.

3.2 Building Blocks

Some constructions for e-cash relies on *blind signatures*. However, we describe a construction using a primitive similar to what is needed for group signatures: signatures with efficient protocols *à la* Camenisch and Lysyanskaya [10].

Definition 12. A *signature with efficient protocols* is a signature scheme (Keygen, Sign, Verify) along with two companion protocols:

- One protocol to prove the knowledge of a signature;
- One protocol to obtain a signature on a committed value.

As this primitive enables a simple anonymous authentication mechanism, it has proven to be a fundamental building block for privacy-preserving cryptography. The construction of signatures with efficient protocols proposed by Camenisch and Lysyanskaya [9] relies on the Strong RSA assumption:

Input: N, a, b, c, and the commitment parameters N_c, g_c, h_c for c.
The user knows r_c such that $c = g^x h^{r_c} \bmod N$.

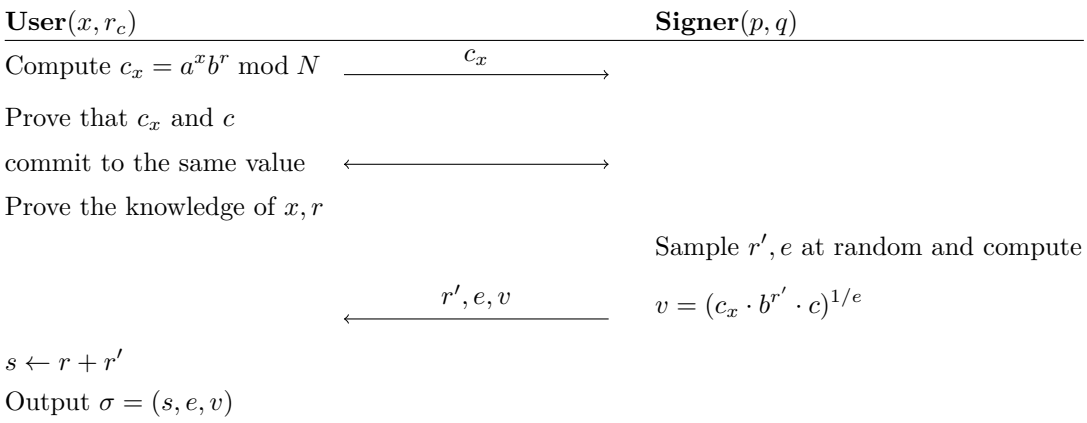

Figure 5: Protocol to sign on a committed value

Definition 13 (Strong RSA)**.** The *strong RSA* problem is the following: given a RSA modulus $N = pq$, and $u \in \mathbb{Z}_N^\star$, find a pair (e, x) such that $u = x^e \bmod N$.

The strong RSA assumption states that no PPT adversary can solve the RSA problem with more than negligible probability.

Keygen(1^λ)**:** sample a *special* RSA-modulus: $N = p \cdot q$ with p and q be safe primes. Sample quadratic residues modulo N: $a, b, c \hookleftarrow QR_N$. Finally, define $\mathsf{pk} = (N, a, b, c)$ and $\mathsf{sk} = p$.

Sign(sk, m)**:** to sign a message from a space made of strings of length ℓ, first pick a random bitstring e of length $\ell + 2$ and s of length $\ell + |p| + \lambda$ and using p compute v such that

$$v^e = a^m b^s c \bmod N. \tag{1}$$

The signature σ consists of the tuple (s, e, v).

Verify(pk, σ, m)**:** to verify a signature σ parsed as (s, e, v), one has to check if equation (1) holds.

Claim. The above signature scheme is EU-CMA-secure under the strong-RSA assumption.

Now that we have a signature scheme, we have to describe the so-called companion-protocols. To do this, we first need to describe the associated commitment scheme, which will be akin to Pedersen's commitment.

Setup(1^λ)**:** generate a special RSA modulus $N = p \cdot q$ and sample $h \hookleftarrow QR_N$ and pick a random g in $\langle h \rangle$ (the group generated by h).

Commit$(x; r)$**:** to commit to a value x using the randomness r, output $\mathsf{com} = g^x h^r \bmod N$ and $\mathsf{dec} = (x, r)$.message

Open$(\mathsf{com}, \mathsf{dec})$**:** to open a commitment com, accept if an only if $\mathsf{com} = g^x h^r \bmod N$.

This commitment is statistically hiding and is computationally binding under the factorisation assumption. Thus, to sign on a committed value, the user and the signer act as described in Figure 5 and the protocol to prove knowledge on a committed value is sketched in Figure 6.

Input: $(N, a, b, c), (g, h), (n_c, g_c, h_c), c_x = g^x \cdot h_c^r$

User$(x, r, \sigma = (s, e, v))$		**Verifier**
Sample randomnesses		
w, r_e, r_w, r_w, r		
Compute		
$c_s \leftarrow g^s h^{r_s} \quad c_e \leftarrow g^e h^{r_e}$		
$c_v \leftarrow g^v h^{r_v} \quad c_w \leftarrow g^w h^{r_w}$		
$z \leftarrow ew \quad c_z = g^z h^{r_z}$	$\xrightarrow{c_s, c_e, c_v, c_w, c_z}$	
Prove relations between commitments	$\longleftrightarrow$	Accepts if the proofs are correct

Figure 6: Proof of Knowledge of a Signature

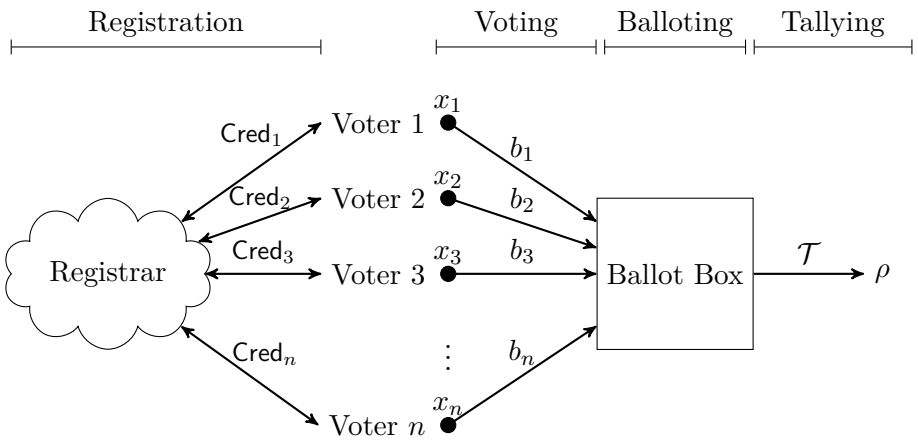

Figure 7: Conduct of an election.

4 Electronic Voting

In this section, we will see yet another complex cryptographic primitive: electronic voting. As its name implies, the goal of electronic voting is to enable a set of user to cast a vote while keeping everything but the result secret. Thus, this protocol is intimately tied to privacy-based cryptography and has been studied since 1985 [14, 4]. Here, we are interested in the case where the vote is made from a computer and is secured over some public channel, such as the internet. We are not considering the case of voting machine where the result is tallied on site.

As anyone can act as they want over the protocol, it should be robust against double-voting attacks, or trying to eavesdrop the result before the tally. As such, the security of the system should protect the users' anonymity, but also the system's soundness to ensure the reliability of the vote.

4.1 Desired Notions

As depicted in Figure 7, a voting system follows a sequence of phases. First, the voters that belongs to the election list receive their credentials from a registrar during the registration phase. Once this is done, a voter casts its vote x_i as its ballot during the voting phase. The

entity responsible of the balloting ensures that public information about the bulletin board remains unlinkable from the ballots. At some point, the vote closes, and the ballots are tallied to obtain the result of the vote $\rho(x_1, \ldots, x_n)$. In a majority vote, a vote is typically a binary vector of Hamming weight at most 1 and the result function is the sum of these vectors. The candidate who obtains the highest number of votes usually wins the election. Each voter wants their vote to remain unlinkable from them (otherwise, they may be subject to coercion), but they may also want to be able to checks that their vote has indeed been taken into account.

Henceforth, many security notions are desirable for an e-voting scheme and we can cite the following properties:

Democracy: this notion captures the fact that every voter should have the same influence on the resulting vote. Meaning that: only eligible voters are allowed to cast a vote, and at most one vote per user should be counted.

Privacy: this notion captures the anonymity of the users under *public ballots*. Hence, no one can tell what lies inside a ballot.

Coercion-resistance: we also want to ensure that no one can be bribed to enforce his/her vote. Thus, a voter shouldn't be able to prove whom it voted for.

Accuracy: the vote count should be exact.

Verifiability: this notion may have multiple variant, but it says that it should be possible to check if a vote is part of the final tally or not. It can be public, representative (only allowed members of the system can check the validity of a vote) or universal (anyone can check that any vote has been casted).

Robustness: the system should be able to resist against a group of malicious voters who wants to corrupt the result of the vote.

Fairness: to ensure the fairness of the vote, we also want that no partial result can be computed before the end of the vote.

Flexibility: a property that does not exist in physical voting is that we may also want the users to be able to change the content of their vote before the end of the vote, so that they can correct a mistake.

These notions, are not all compatible with each others. For example, a system cannot be both individually verifiable and resistant to coercion, as the verification of your own vote is a proof of what you voted for.

As such, the description of a voting system will depend on the security it should ensure. For instance the *Belenios* voting system [15], which is individually verifiable, involves the following entities:

Election administrator: this entity $\mathcal{E}$ is responsible for setting up an election. It publishes the identities id of eligible voters, the list of candidates and the result function ρ of the election.

Registrar: this entity $\mathcal{R}$ is responsible for distributing secret credentials to voters and registering the corresponding public credentials.

Trustee: this entity $\mathcal{T}$ is in charge of tallying and publishing the final result.

Voters: the eligible voters, denoted by their id $id_1, \ldots, id_n$, who participate in the election.

Bulletin board manager: denoted by $\mathcal{B}$, this entity is responsible for processing ballots and storing the valid ballots in the bulletin board BB.

These entities participate in the vote via different algorithms. The voting protocol is defined by the set of admissible vote $\mathbb{V}$, the result space $\mathbf{R}$ and the result function family $\{\rho_n : \mathbb{V}^n \to \mathbf{R}\}_{n\in\mathbb{N}}$. In the case of a majority vote, $\mathbb{V}$ is the set of binary vectors (indexed by the candidates) with at most one 1 and the result space is an integer vector, the result function is just the vector sum function over casted votes. With these parameters, the voting system consists of the following algorithms:

$\mathsf{Setup}(1^\lambda)$: from the security parameter λ, this algorithm outputs an election key pair $(\mathsf{pk}, \mathsf{sk})$ and a list of credentials L. The public key pk is implicit in all the following algorithms.

$\mathsf{Credential}(1^\lambda, \mathsf{id})$: from the security parameter λ and some user identity id, this algorithm generates the user's credential $(\mathsf{pk}_{\mathsf{id}}, \mathsf{sk}_{\mathsf{id}})$ and adds $\mathsf{pk}_{\mathsf{id}}$ to the list of credentials L.

$\mathsf{Vote}(\mathsf{id}, \mathsf{pk}_{\mathsf{id}}, \mathsf{sk}_{\mathsf{id}}, v)$: each user can cast a vote using its credentials and its vote $v \in \mathbb{V}$. It outputs a ballot b.

$\mathsf{Validate}(b)$: on input a ballot b, this algorithm returns $\top$ for well-formed ballot or $\bot$ otherwise.

$\mathsf{Box}(\mathsf{BB}, b)$: from the bulletin board BB and a ballot b, this algorithm outputs an updated bulletin board BB'. This algorithm may be stateful (meaning that it maintains a local state st). In any cases, BB remains unchanged if $\mathsf{Validate}(b) = \bot$. We say that BB is well-formed if $\mathsf{Validate}(b) = 1$ for every $b \in \mathsf{BB}$.

$\mathsf{VerifyVote}(\mathsf{BB}, \mathsf{id}, \mathsf{pk}_{\mathsf{id}}, \mathsf{sk}_{\mathsf{id}}, b)$: this algorithm returns $\bot$ or $\top$ from the bulletin board BB, the voter's credentials, and a ballot b.

$\mathsf{Tally}(\mathsf{BB}, \mathsf{sk})$: from the bulletin board BB and the election secret key sk, this algorithm outputs the tally r and a proof of correct tabulation Π. It is possible that Π is equal to $\bot$ which corresponds to an invalid election.

$\mathsf{Verify}(\mathsf{BB}, r, \Pi)$: from the bulletin board BB, the result r and the proof of correct tabulation Π, this algorithm returns $\bot$ or $\top$.

4.2 Designing a Voting System

Several methods have been proposed to design an e-voting system that rely on different building blocks.

From homomorphic encryption, it is possible to obtain universal verifiability but the votes are not flexible. Using blind signatures to ensure the anonymity of the votes allows self-verification. Finally, a last solution uses mix networks that are defined below.

Definition 14 (Mix Network [13])**.** A *mix-network* (or *mix-net*) with respect to a public key encryption PKE is a function Mix such that $\mathsf{Mix}(C_1, \ldots, C_n) = (C'_{\sigma(1)}, \ldots, C'_{\sigma(n)})$ where σ is a random permutation and $\mathsf{PKE.Dec}_{\mathsf{sk}}(C'_{\sigma(i)}) = \mathsf{PKE.Dec}_{\mathsf{sk}}(C_{\sigma(i)})$ for any $i \in [1, n]$.

When this function is associated with a zero-knowledge proof that allows to prove a correct shuffling of the encrypted inputs, the mix-network is said to be *verifiable*.

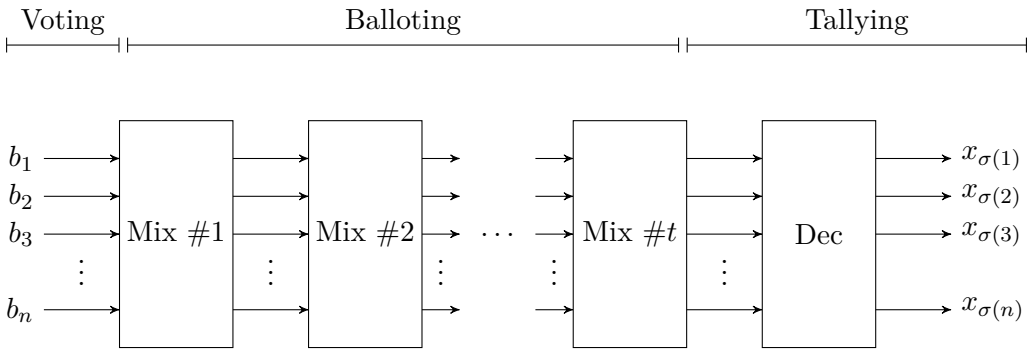

Figure 8: Mix-network-based voting system.

Mix-net-based solutions allow both flexibility and universal-verifiability, however fairness is an issue. However these schemes are quite heavy in terms of communication and computation costs. To design a voting scheme from mix-networks, the ballots (which are encrypted votes) are put into a series of mix-nets and the resulting vector (along with the proof of correct shuffling). Thus the tallying is just the decryption of the resulting vector along with a proof of correct decryption, and the result function is applied to the resulting results. The use of mix-networks ensure that voter cannot be linked back to voters and zero-knowledge proofs ensure verifiability and robustness of the scheme. We can also see that modifying a vote at the input of the mix-nets layers is possible with this scheme.

Homomorphic encryptions allows to simply encrypt the vote and aggregates them in the bulletin board (with a proof of correct evaluation) to make impossible to find back a vote from the (publicly available) bulletin board. Then, to tally the vote, one simply has to decrypt the bulletin board.

Finally, for blind signature-based e-voting systems, the vote procedure is an interactive protocol with the election administrator. The voter blindly signs its vote to obtain a signature σ_i and sends its ballot $b = \mathsf{PKE.Enc}(\mathsf{pk}, \langle x_i, \sigma_i \rangle)$ to the trustee. In the original scheme from Fujioka, Okamoto and Ohta [18], the fairness is ensured by splitting the trustee in two entities: an opener and a counter. In this case, the voter sends an encryption of a commitment its vote to the opener and the decommitment of its vote to the counter. Therefore, the counter cannot count the votes without the opener first opening them, and the opener can't do anything from the commitment of the votes (from the hiding property). The blind signature ensures both anonymity and along with the binding of the commitment the robustness of the system.

We can remark that in any of those schemes, it is possible to split the different authorities to avoid a single point of failure using secure multiparty computation (MPC). It is what Belenios is doing for instance.

4.3 Instantiating an E-Voting Scheme: Belenios

Belenios [15] is a e-voting scheme that derives from Helios [1]. It works using the homomorphism of the additive variant of El Gamal encryption and the Schnorr signature scheme from discrete logarithm (which is the Fiat-Shamir instanciation of the Schnorr identification protocol we saw previously). The zero-knowledge proof that is used is a Schnorr-like proof.

Setup(1^λ)**:** this algorithm generates the public parameters and a key pair for El Gamal encryption. Namely, it picks $\mathbb{G} = \langle g \rangle$ a cyclic group of order $p > 2^\lambda$, and sets $\mathsf{sk} \hookleftarrow \mathbb{Z}_p$ and $h = g^{\mathsf{sk}}$. It also samples two hash functions $H, G : \{0,1\}^\star \to \mathbb{Z}_p$, initialises a credentials list L as $\emptyset$ and defines the public key as $\mathsf{pk} \triangleq (\mathbb{G}, p, h, L, G, H, \mathbb{V} = \{0,1\})$.

Credential$(1^\lambda, \mathsf{id})$**:** It generates a signing key for the voter who asks for it $(\mathsf{pk}_{\mathsf{id}}, \mathsf{sk}_{\mathsf{id}}) \leftarrow \mathsf{Sig.Keygen}(1^\lambda)$ and appends $\mathsf{pk}_{\mathsf{id}}$ to L.

Vote$(\mathsf{id}, \mathsf{pk}_{\mathsf{id}}, \mathsf{sk}_{\mathsf{id}}, v)$**:** To cast its vote using its credentials, a user generates its ballot b as follows:

1. it encrypts its vote v as $c = (c_1, c_2) \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, g^v; r)$ and computes an OR-proof π using $g, \mathsf{pk}, c_1, c_2, r$ that v encrypts either 0 or 1 (using the hash function H).
2. it computes $\sigma \leftarrow \mathsf{Sig.Sign}(\mathsf{pk}_{\mathsf{id}}, (c, \pi))$ and defines the ballot as $b = (\mathsf{pk}_{\mathsf{id}}, c, \pi, \sigma)$.

Validate(b)**:** to validate a ballot, the proof π is verified as well as the signature σ with respect to $\mathsf{pk}_{\mathsf{id}}$. If either of those check fails it outputs $\perp$ otherwise it returns $\top$.

Box(BB, b)**:** this algorithm maintains a local state st that stores entries of the form $(\mathsf{id}, \mathsf{pk}_{\mathsf{id}})$. If $\mathsf{Validate}(b) = \top$, and the voter id never casted a vote before, it updates $\mathsf{BB} \leftarrow \mathsf{BB} \cup \{b\}$ and updates st accordingly. If $(\mathsf{id}, \mathsf{pk}_{\mathsf{id}})$ already appears in its recorded state, it updates the corresponding entry in BB. In any other cases, it means that the casted vote is invalid and BB remains the same.

VerifyVote$(\mathsf{BB}, \mathsf{id}, \mathsf{pk}_{\mathsf{id}}, \mathsf{sk}_{\mathsf{id}}, b)$**:** to verify a vote, this algorithm merely checks that $b \in \mathsf{BB}$.

Tally$(\mathsf{BB}, \mathsf{sk})$**:** when it's time to tally, this algorithm performs the following steps:

1. For each $b \in \mathsf{BB}$, run $\mathsf{Validate}(b)$. If any of these verifications return $\perp$, return $\perp$ and $\Pi = \perp$.
2. Parse any ballot b as $(\mathsf{pk}_b, c, \pi, \sigma)$.
3. If pk_b appears twice in BB or $\mathsf{pk}_b \notin L$, it outputs $\perp$ and $\Pi = \perp$.
4. Compute the result ciphertext $c_\Sigma = (c_1^\Sigma, c_2^\Sigma) = (\prod_{b \in \mathsf{BB}} c_1^b, \prod_{b \in \mathsf{BB}} c_2^b)$ where c_b is (c_1^b, c_2^b).
5. Decrypt c_Σ to obtains g^ρ and runs a generic discrete log algorithm (for instance baby-step-giant-step) to retrieve the result ρ in time $O(\sqrt{n})$ where n is the number of legitimate voters.
6. Finally, Π is the proof that c_Σ encrypts g^ρ (using the hash function G), and outputs (ρ, Π).

Verify(BB, r, Π)**:** to check the validity of a tally, this algorithm runs the verifications steps 1-3 of the Tally algorithm, then recompute the result ciphertext c_Σ as in step 4 of the tally algorithm and checks the proof Π that c_Σ is indeed an encryption of g^ρ.

This voting scheme is proven secure in the random oracle model with *weak verifiability* (meaning that it enjoys universal and individual verifiability under honest bulletin board and registration authorities). However, a generic transform exist to make it secure if one of these two is malicious (but not both).

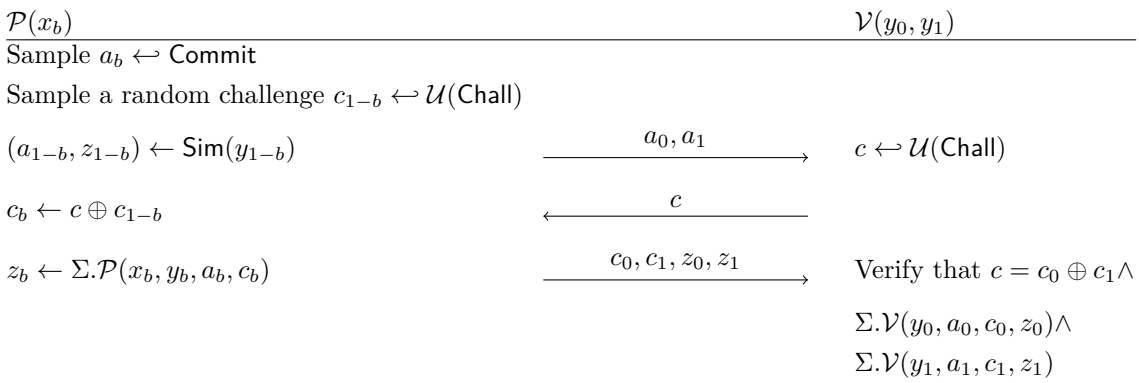

Figure 9: Description of an OR-proof

OR PROOFS. Given two NP-statements (x_0, y_0) and (x_1, y_1) and a $\Sigma-$protocol Σ to prove the knowledge of a witness x for a word y in the language. Let us design a Σ-protocol such that the prover shows that it knows either x_0 or x_1. The description of the construction is given in Figure 9.

References

[1] B. Adida. Helios: Web-based open-audit voting. In *USENIX security symposium*, volume 17, pages 335–348, 2008.

[2] M. Bellare, D. Micciancio, and B. Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In *Eurocrypt*, volume 3376 of *LNCS*, pages 614–629. Springer, 2003.

[3] M. Bellare, H. Shi, and C. Zhang. Foundations of group signatures: The case of dynamic groups. In *CT-RSA*, volume 2656 of *LNCS*, pages 136–153. Springer, 2005.

[4] J. C. Benaloh and M. Yung. Distributing the power of a government to enhance the privacy of voters. In *Proceedings of the fifth annual ACM symposium on Principles of distributed computing*, pages 52–62, 1986.

[5] D. Boneh and M. Franklin. Identity-based encryption from the weil pairing. In J. Kilian, editor, *Crypto*, pages 213–229. Springer, 2001.

[6] J. Camenisch, S. Hohenberger, and A. Lysyanskaya. Compact e-cash. In *Eurocrypt*, number 3494 in LNCS, pages 302–321. Springer, 2005.

[7] J. Camenisch and A. Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *Eurocrypt*, number 2045 in LNCS, pages 93–118. Springer, 2001.

[8] J. Camenisch and A. Lysyanskaya. A signature scheme with efficient protocols. In *SCN*, number 2576 in LNCS, pages 268–289. Springer, 2002.

[9] J. Camenisch and A. Lysyanskaya. A signature scheme with efficient protocols. In *Security and Cryptography for Networks (SCN'02)*, number 2576 in LNCS, pages 268–289, 2002.

[10] J. Camenisch and A. Lysyanskaya. A Signature Scheme with Efficient Protocols. In *SCN*, LNCS, pages 268–289. Springer, 2004.

[11] D. Chaum. Security without Identification: Transactions System to Make Big Brother Obsolete. 28(10):1030–1044, 1985.

[12] D. Chaum and E. van Heyst. Group signatures. In *Eurocrypt*, volume 547 of *LNCS*, pages 257–265. Springer, Springer, 1991.

[13] D. L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–90, feb 1981.

[14] J. D. Cohen and M. J. Fischer. A robust and verifiable cryptographically secure election scheme. In *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*, pages 372–382. IEEE Computer Society, 1985.

[15] V. Cortier, D. Galindo, S. Glondu, and M. Izabachene. Election verifiability for helios under weaker trust assumptions. In *European Symposium on Research in Computer Security*, pages 327–344. Springer, 2014.

[16] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

[17] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In A. M. Odlyzko, editor, *Crypto*, pages 186–194. Springer, 1986.

[18] A. Fujioka, T. Okamoto, and K. Ohta. A practical secret voting scheme for large scale elections. In *International Workshop on the Theory and Application of Cryptographic Techniques*, pages 244–251. Springer, 1992.

[19] A. Kiayias and M. Yung. Group signatures with efficient concurrent join. In *Eurocrypt*, number 3494 in LNCS, pages 198–214. Springer, 2005.

[20] A. Kiayias and M. Yung. Secure scalable group signature with dynamic joins and separable authorities. 1(1):24–45, 2006.

[21] B. Libert, S. Ling, F. Mouhartem, K. Nguyen, and H. Wang. Signature schemes with efficient protocols and dynamic group signatures from lattice assumptions. In *Asiacrypt*, 2016.

[22] B. Libert, F. Mouhartem, T. Peters, and M. Yung. Practical "signatures with efficient protocols" from simple assumptions. In *AsiaCCS*, pages 511–522. ACM, 2016.

[23] B. Libert, T. Peters, M. Joye, and M. Yung. Linearly Homomorphic Structure-Preserving Signatures and Their Applications. In *Crypto*, LNCS, pages 289–307. Springer, 2013.

[24] B. Libert, T. Peters, and M. Yung. Short group signatures via structure-preserving signatures: Standard model security from simple assumptions. In *Crypto*, volume 9216 of *LNCS*, pages 296–316. Springer, 2015.

[25] T. P. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *Crypto*, pages 129–140. Springer, 1991.

[26] D. Pointcheval and J. Stern. Security Arguments for Digital Signatures and Blind Signatures. *Journal of Cryptology*, 13(3):361–396, June 2000.

[27] Y. Sakai, K. Emura, G. Hanaoka, Y. Kawai, T. Matsuda, and K. Omote. Group signatures with message-dependent opening. In *Pairing'12*, volume 7708 of *LNCS*, pages 270–294. Springer, 2012.

[28] C. P. Schnorr. Efficient identification and signatures for smart cards. In *Crypto*, volume 435 of *LNCS*, pages 239–252, 1989.