# Privacy-Preserving User-Auditable Pseudonym Systems

Jan Camenisch
IBM Research – Zurich
jca@zurich.ibm.com

Anja Lehmann
IBM Research – Zurich
anj@zurich.ibm.com

*Abstract*—**Personal information is often gathered and processed in a decentralized fashion. Examples include health records and governmental data bases. To protect the privacy of individuals, no unique user identifier should be used across the different databases. At the same time, the utility of the distributed information needs to be preserved which requires that it be nevertheless possible to link different records if they relate to the same user. Recently, Camenisch and Lehmann (CCS 15) have proposed a pseudonym scheme that addresses this problem by domain-specific pseudonyms. Although being unlinkable, these pseudonyms can be converted by a central authority (the converter). To protect the users' privacy, conversions are done blindly without the converter learning the pseudonyms or the identity of the user. Unfortunately, their scheme sacrifices a crucial privacy feature: transparency. Users are no longer able to inquire with the converter and audit the flow of their personal data. Indeed, such auditability appears to be diametral to the goal of blind pseudonym conversion. In this paper we address these seemingly conflicting requirements and provide a system where user-centric audits logs are created by the oblivious converter while maintaining all privacy properties. We prove our protocol to be UC-secure and give an efficient instantiation using novel building blocks.**

## 1. Introduction

In our forming digital society, personal data is increasingly collected, processed, maintained, and exchanged in electronic form. When data collection and operations happen in a distributed fashion, it is often important that different data sets of the same user can be associated. Typical examples of such distributed, yet linkable data sets are health records or governmental databases. Many countries, including the US, Belgium, Denmark, and Sweden, use a nation-wide social security number for linkage. While the use of such unique identifiers across the whole system easily allows the different entities to correlate their records, it poses serious risks to data security and user privacy. For one, it is hard if not impossible to control and limit the exchange of records between entities. Even worse, any data breach reveals fully identifiable and linkable personal information. Because this is a major threat to user privacy and a security risk, such databases require high protection, which is often very costly.

A better approach is to use entity-specific identifiers (called *pseudonyms*) which per se are unlinkable. To still allow for linkage of a user's different records, these pseudonyms are established via a central entity (the *converter*), who can later be asked to convert between pseudonyms of the same user on a case per case basis [14]. In Japan, a new social security and tax number system, called "MyNumber," following that approach has been rolled-out in 2015 [22]. Also the recent eID system in Austria [11] allows (in theory) for such convertible pseudonyms. A pseudonym system with such a central converter clearly has strong advantages regarding privacy of the data records and controllability of the information flow. However, as the converter is necessarily involved in every data exchange and learns which entities want to correlate data of which user, this central authority is very powerful, must be fully trusted, and becomes a target of attack. To avoid a privacy and security nightmare, the converter needs to be very well protected and still be highly available – an almost impossible task.

A better approach that addresses this problem by preventing the converter from learning all this information has recently been proposed by Camenisch and Lehmann [7]. In their system, termed the CL-15 system henceforth, the converter is still the crucial entity to establish and to translate between the (un)linkable pseudonyms, but it does so in a blind way, i.e., without learning the pseudonyms or identity of the user for whom a conversion is requested. The converter cannot even tell whether or not two requests relate to the same user. The only information it learns is that one server, say $\mathcal{S}_A$, wants to access some related data from another server, say $\mathcal{S}_B$. This allows the converter to impose access control based on the servers' identities.

*Transparency vs. Privacy.* While in terms of privacy the CL-15 system is clearly superior to the other approaches discussed, it misses one very crucial feature: transparency and auditability. Indeed, a user is not able to follow and audit the flow of his personal data. Transparency is vital for compliance with legal and business requirements and has been recognized as one of the three key principles in the European Data Protection Directive. It states that data subjects have the right to be informed when their personal data are being processed. In fact, in the new social security number system of Japan, where entity-specific pseudonyms are used, a portal giving users access to the exchange history of their data has been a crucial design requirement. A realization called "MyPortal" is supposed to be launched in 2017 [22].

Providing transparency is rather trivial in a pseudonym system with a converter who learns the identity of the user behind every request: the converter can simply create an audit log for each data request and inform the users of all exchanges of their personal data. A number of authors discuss different mechanisms to publish such audit logs in a secure and privacy-friendly manner, ensuring, e.g., that personal information in audit logs is only accessible to the respective user or that logs do not contain linkable traces [21], [20], [26]. All these audit solutions crucially rely on the converter's knowledge of the underlying users' identities. However, this information is exactly what the CL-15 system hides from all involved entities. In fact, allowing the converter (or the servers) to generate user-specific information, such as an entry in an audit log, seems to inherently contradict the blindness property that guarantees that neither the converter nor the servers can learn anything about a user's identity when generating or converting pseudonyms.

A possible attempt to add transparency to the CL-15 system would be to introduce a dedicated audit entity $\mathcal{S}_T$ and to extend the pseudonym conversion protocol as follows. Whenever a server $\mathcal{S}_A$ wants to convert a pseudonym $nym_{i,A}$ towards $\mathcal{S}_B$, the converter also blindly transforms $nym_{i,A}$ into a pseudonym $nym_{i,T}$ towards the auditor $\mathcal{S}_T$, and hands the conversion context to $\mathcal{S}_T$. As all pseudonyms of the underlying user $\mathcal{U}_i$ will be consistently transformed into $nym_{i,T}$, the auditor is able to maintain the entire conversion history for each user. Indeed, such a logging approach was recently proposed by Verheul et al. [23], where a pseudonym system with (semi)blind conversions is described that is currently deployed in a medical research project. While such an extension adds the desired auditability, it destroys privacy: the auditor learns of all related conversion requests and thus must be fully trusted and well protected. Indeed, such a solution offers no advantage over a solution where the converter is aware of the users' identities and creates the logs, in particular if users are allowed online access to their logs.

*Our Contributions.* In this paper we show that transparency and privacy are not contradictory by presenting an efficient pseudonym system with build-in auditability that enjoys the same strong privacy properties as the CL-15 system. Conceptually, we aim at a similar audit setup as in a pseudonym system with a fully trusted converter: the converter creates an audit log entry upon each conversion request and makes these logs publicly available. This is done such that users can retrieve their log entries and inform themselves about the exchange of their own data. Despite the added auditability, no privacy is lost: all pseudonym conversions and generations are fully unlinkable and do not reveal any information about the underlying user to the converter. Indeed, only the user to whom different log entries relate is able to link them.

While from a system perspective, the addition of audit capabilities might seem like a rather small extension, realizing it in a privacy-preserving and provably-secure manner is far from trivial. The main idea behind our construction is as follows: we associate each blindly generated pseudonym with a public encryption key so that the underlying user knows the corresponding decryption key. That public key is carried along whenever a pseudonym is converted. However, to retain the privacy of conversions, the public key must be randomized. To enable this, we formally define a new kind of encryption scheme that allows for such *key randomization* and show an instantiation based on ElGamal. A similar concept has been used to obtain encryption schemes with receiver anonymity [24], [27].

The core of our pseudonym conversion protocol is similar to the one in the CL-15 scheme. When a server $\mathcal{S}_A$ wishes to convert the pseudonym $nym_{i,A}$ (of the underlying user $\mathcal{U}_i$) towards a server $\mathcal{S}_B$, it sends the pseudonym in homomorphically encrypted form to the converter, who then blindly computes $nym_{i,B}$ by performing the conversion homomorphically on the ciphertext. In our scheme, the encrypted pseudonym is accompanied by a freshly randomized encryption key of the related, yet unknown user. The converter then encrypts under this randomized key the audit information that records the data exchange between $\mathcal{S}_A$ and $\mathcal{S}_B$. The encrypted audit logs are made publicly available such that they can be fetched by all users, yet users can only decrypt *their own* logs entries.

This basic scheme just sketched already provides both auditability and privacy, but only if all entities follow the protocols honestly. Given that a pseudonym system is designed for a distributed environment, where a multitude of servers and users with diverse and hard to control backgrounds participate, assuming all parties to be honest-but-curious is far too strong an assumption. Thus, to ensure that malicious users or servers cannot deviate from the protocol, we deploy a new type of signature scheme, i.e., one for signing encoded messages. Such a scheme allows the converter to blindly sign tuples of transformed and encrypted pseudonyms and randomized user public keys, and then to later blindly verify that servers only use legitimate combinations of pseudonyms and (randomized) keys. This new kind of signature scheme has many other applications and is considered a result of separate interest. We also describe a realization based on a recent structure-preserving signature scheme by Groth [15].

Ensuring security against a fully corrupted converter would require a substantial overhead in form of distributed consensus protocols and a number of additional zero-knowledge proofs. By assuming that a corrupt converter will still perform the protocol to specification (i.e., the converter to be honest-but-curious), we can avoid this complexity yet cover the main threat: an overly curious converter who tries to trace or identify users by exploiting all the information and keys it sees. We believe this is a reasonable setting. First, because the converter performs operations blindly, its correct behaviour can be tested at any time by dummy conversions for which the servers know the outcome. Second, because it being a central and important entity, its correct operation can further be ensured for instance by using trusted hardware and auditing mechanisms.

Another drawback of the basic scheme sketched above is that users need to attempt the decryption of *all* published records which is far from practical in systems used on a nation-wide level. We therefore introduce a tag-chaining approach so that user will be able to recognize their entries. This requires that users be involved in the pseudonym generation protocol (but still not in the conversion protocol), so that they can obtain their initial tags. This involvement of the user is different from the CL-15 system [7], where the pseudonym *generation* was always triggered by the converter and it was left outside the model how the converter learns the identity of the user for whom to generate the pseudonym. The involvement of the user actually allows us to even increase the privacy properties of the pseudonym system. CL-15 only guarantees privacy and unlinkability for pseudonym conversion, but not for the generation where the converter is always privy of the user behind a pseudonym request. In our scheme, the converter blindly computes the pseudonym without learning the user's identity. This is achieved using a new *committed* oblivious pseudorandom function evaluation protocol, which we define formally and instantiate securely.

We prove the security of our generic construction in the UC framework by showing that it realizes the ideal functionality $\mathcal{F}_{\text{nym-log}}$. This functionality is a modification of the CL-15 one, extended to cover blind pseudonym generation and user-centric auditability. We also give concrete and optimized instantiations for all building blocks, yielding an efficient realization of our scheme under discrete-logarithm based assumptions and Paillier's DCR assumption.

## 2. Security Model

In this section we provide the formal definition of our (un)linkable pseudonym system with user auditability. Our definition is based on the ideal functionality given by Camenisch and Lehmann [7], which we modify to enhance the user privacy for pseudonym generation and allow the user to monitor the conversions of his pseudonyms. We start by recalling the main entities and procedures of an (un)linkable pseudonym system, and then present our formal definition and discuss the properties it guarantees.

The main entities in an auditable pseudonym system are a converter $\mathcal{X}$, a set of servers $\mathbf{S} = \{\mathcal{S}_A, \mathcal{S}_B, \dots\}$ and users $\mathcal{U}_i$. Explicitly modeling the user as part of the system is already the first crucial difference to the CL-15 model, where the user was only represented by a user identifier that was known to the converter.

The converter $\mathcal{X}$ is the central authority that must be involved in any generation and conversion of the (un)linkable pseudonyms. The generation of a pseudonym $nym_{i,A}$ for user $\mathcal{U}_i$ on server $\mathcal{S}_A$ is initiated by the user towards the converter. Upon explicit approval of the converter, the server-specific pseudonym $nym_{i,A}$ gets established and output directly to the server $\mathcal{S}_A$, i.e., without $\mathcal{X}$ learning the pseudonym. Further, neither $\mathcal{S}_A$ nor $\mathcal{X}$ learn the user behind the pseudonym which is different to CL-15 where the user was (in pseudonym generation) always known to the converter.

The converter $\mathcal{X}$ is the only entity in the system that can link related pseudonyms $nym_{i,A}$ and $nym_{i,B}$ of the same user, but will do so in a blind manner, i.e., without learning anything about the pseudonyms in the request. A conversion is initiated by $\mathcal{S}_A$ towards $\mathcal{X}$ whenever the server wishes to correlate his data for the user known to him as $nym_{i,A}$ with related user data held by another server $\mathcal{S}_B$. If a conversion is granted by $\mathcal{X}$, only $\mathcal{S}_B$ will learn the converted pseudonym $nym_{i,B}$.

Our main extension is that each conversion now triggers the blind generation of an audit log entry that allows the user behind the pseudonym to learn that $\mathcal{S}_A$ wishes to access his data from $\mathcal{S}_B$. Such an audit log entry is only accessible to the correct underlying user $\mathcal{U}_i$, who can retrieve all his log entries to monitor the propagation of his pseudonyms.

As in CL-15, all pseudonyms are generated and converted in a consistent way, meaning that each user can generate only a single and unique pseudonym per server and that all conversions among servers are transitive and consistent with each other.

*Corruption Model.* We consider two different corruption types: For servers and users we allow active corruptions by the adversary, i.e., upon corruption the adversary is in full control over the servers' or users' behaviour. The converter can be corrupted only in a passive manner, which is also called honest-but-curious model. That is, when corrupted, the adversary sees all of the converter's input, output and internal state, but the converter's behaviour remains honest.

### 2.1. Ideal Functionality

We now formally define our auditable pseudonym system by describing an ideal functionality in the universal composability (UC) framework [10]. The ideal functionality performs the desired task in a way that is secure-by-design. A real-world protocol is then said to securely realize a certain ideal functionality $\mathcal{F}$, if an environment cannot distinguish whether it is interacting with the real protocol or with $\mathcal{F}$ and a simulator.

We assume static corruptions in our paper, i.e., the adversary decides upfront which parties are corrupt and makes this information known to the functionality. The UC framework allows us to focus our analysis on a single protocol instance with a globally unique session identifier $sid$. Here we use session identifiers of the form $sid = (sid', \mathcal{X}, \mathbf{S}, \mathbf{N})$, for some converter and server identifiers $\mathcal{X}, \mathbf{S} = \{\mathcal{S}_A, \mathcal{S}_B, \dots\}$, a unique string $sid'$, and $\mathbf{N}$ denoting the pseudonym space. To distinguish between several pseudonym generation and conversion sessions, we use unique query identifiers $nqid$ and $cqid = (cqid', \mathcal{S}_A, \mathcal{S}_B)$ for each session. We implicitly assume that the functionality checks that all session and query identifiers are well-formed and the query identifiers $nqid, cqid$ are unique.

The definition of our ideal functionality $\mathcal{F}_{\text{nym-log}}$ is presented in detail in Figure 1. For simplicity, we refer to

1) `Pseudonym Request.` On input of $(\mathsf{NYMREQ}, sid, nqid, \mathcal{S}_A)$ from user $\mathcal{U}_i$:

- If $\mathcal{X}$ and $\mathcal{S}_A$ are corrupt, get $\ell \leftarrow \mathsf{leak}(\mathcal{U}_i)$. Otherwise, if at least $\mathcal{X}$ or $\mathcal{S}_A$ is honest, set $\ell \leftarrow \perp$.
- Send $(\mathsf{NYMREQ}, sid, nqid, \mathcal{S}_A, \ell)$ to $\mathcal{A}$ and wait for $(\mathsf{NYMREQ}, sid, nqid, \mathtt{ok})$ from $\mathcal{A}$.
- Create a pseudonym request record as $(\mathsf{nymreq}, sid, nqid, \mathcal{U}_i, \mathcal{S}_A)$.
- Output $(\mathsf{NYMREQ}, sid, nqid, \mathcal{S}_A)$ to $\mathcal{X}$.

2) `Pseudonym Generation.` On input of $(\mathsf{NYMGEN}, sid, nqid)$ from converter $\mathcal{X}$:

- Proceed only if a pseudonym request record $(\mathsf{nymreq}, sid, nqid, \mathcal{U}_i, \mathcal{S}_A)$ for $nqid$ exists.
- Send $(\mathsf{NYMGEN}, sid, nqid)$ to $\mathcal{A}$ and wait for $(\mathsf{NYMGEN}, sid, nqid, \mathtt{ok})$ from $\mathcal{A}$.
- If a pseudonym record $(\mathsf{nym}, sid, \mathcal{U}_i, \mathcal{S}_A, nym_{i,A})$ for $\mathcal{U}_i, \mathcal{S}_A$ exists, retrieve $nym_{i,A}$; else create a new record with $nym_{i,A} \xleftarrow{\$} \mathbf{N}$.
- Output $(\mathsf{NYMGEN}, sid, nqid, nym_{i,A})$ to $\mathcal{S}_A$.

---

3) `Conversion Request.` On input of $(\mathsf{CONVERT}, sid, cqid, nym_{i,A})$ from server $\mathcal{S}_A$:

- Proceed only if a pseudonym record $(\mathsf{nym}, sid, \mathcal{U}_i, \mathcal{S}_A, nym_{i,A})$ for $nym_{i,A}, \mathcal{S}_A$ exists and $cqid = (cqid', \mathcal{S}_A, \mathcal{S}_B)$.
- If $\mathcal{X}$ and $\mathcal{S}_B$ are corrupt, get $\ell \leftarrow \mathsf{leak}(\mathcal{U}_i)$. Otherwise, if at least $\mathcal{X}$ or $\mathcal{S}_B$ is honest, set $\ell \leftarrow \perp$.
- Send $(\mathsf{CONVERT}, sid, cqid, \ell)$ to $\mathcal{A}$ and wait for response $(\mathsf{CONVERT}, sid, cqid, \mathtt{ok})$.
- Create a conversion record $(\mathsf{convert}, sid, cqid, \mathcal{U}_i, status)$ with $status \leftarrow \mathtt{request}$ and with $\mathcal{U}_i$ taken from the pseudonym record $(\mathsf{nym}, sid, \mathcal{U}_i, \mathcal{S}_A, nym_{i,A})$ for the requested $nym_{i,A}, \mathcal{S}_A$.
- Output $(\mathsf{CONVERT}, sid, cqid)$ to $\mathcal{X}$.

4) `Conversion Response.` On input of $(\mathsf{PROCEED}, sid, cqid)$ from converter $\mathcal{X}$:

- Proceed only if a conversion record $(\mathsf{convert}, sid, cqid, \mathcal{U}_i, status)$ for $cqid$ with $status = \mathtt{request}$ exists, set $status \leftarrow \mathtt{done}$.
- Send $(\mathsf{PROCEED}, sid, cqid)$ to $\mathcal{A}$ and wait for $(\mathsf{PROCEED}, sid, cqid, \mathtt{ok})$ from $\mathcal{A}$.
- If a pseudonym record $(\mathsf{nym}, sid, \mathcal{U}_i, \mathcal{S}_B, nym_{i,B})$ for $\mathcal{U}_i, \mathcal{S}_B$ exists (with $\mathcal{S}_B$ taken from $cqid = (cqid', \mathcal{S}_A, \mathcal{S}_B)$), retrieve $nym_{i,B}$; otherwise create a new pseudonym record with $nym_{i,B} \xleftarrow{\$} \mathbf{N}$.
- Output $(\mathsf{CONVERTED}, sid, cqid, nym_{i,B})$ to $\mathcal{S}_B$.

---

5) `User Audit.` On input of $(\mathsf{AUDIT}, sid)$ from user $\mathcal{U}_i$:

- Retrieve all conversion records $(\mathsf{convert}, sid, cqid_j, \mathcal{U}_i, status_j)$ for $\mathcal{U}_i$ and assemble a list $\mathbf{L}_{\text{log}} \leftarrow \{(cqid_j, status_j)\}$ from all found records. Recall that $cqid_j = (cqid'_j, \mathcal{S}_A, \mathcal{S}_B)$.
- Output $(\mathsf{AUDIT}, sid, \mathbf{L}_{\text{log}})$ to $\mathcal{U}_i$.

Figure 1: Ideal functionality $\mathcal{F}^{\mathsf{leak}}_{\text{nym-log}}$ with $sid = (sid', \mathcal{X}, \mathbf{S}, \mathbf{N})$ and parametrized with a leakage function $\mathsf{leak}$.

$\mathcal{F}_{\text{nym-log}}$ as $\mathcal{F}$ from now on. In the following we informally discuss the security properties that our functionality provides. Thereby we focus on the audit related procedures, for a detailed discussion of the privacy and security properties of the basic pseudonym system we refer to [7].

*Pseudonym Generation.* The NYMREQ interface allows a user $\mathcal{U}_i$ to request the generation of a pseudonym $nym_{i,A}$ towards server $\mathcal{S}_A$. The converter $\mathcal{X}$ is then informed about that request but without learning the user identity $\mathcal{U}_i$. When both $\mathcal{X}$ and $\mathcal{S}_A$ are corrupt though, the adversary will receive some partial information of $\mathcal{U}_i$ via $\ell \leftarrow \mathsf{leak}(\mathcal{U}_i)$. We discuss the leakage function and its relation to the CL-15 model in more detail below.

When the converter approves the pseudonym generation via the NYMGEN interface, $\mathcal{F}$ either retrieves the pseudonym $nym_{i,A}$ for $(\mathcal{U}_i, \mathcal{S}_A)$ from its internal record or creates a new one. Every pseudonym is chosen at random from $\mathbf{N}$ which naturally enforces unlinkability of pseudonyms. The functionality internally stores each pseudonym in a record that connects $\mathcal{U}_i$ and $nym_{i,A}$ which allows for consistent conversion and the user-specific audits. Finally, the pseudonym $nym_{i,A}$ is then output directly to $\mathcal{S}_A$.

As $\mathcal{X}$ no longer knows the user's identity in a pseudonym generation, we omit the verifiability of pseudonyms that was provided in CL-15. Instead we associate each pseudonym generation session with a unique session identifier $nqid$, that is given to $\mathcal{U}_i, \mathcal{X}$ and $\mathcal{S}_A$ and that can be used by $\mathcal{U}_i$ to provide some context to the blindly generated $nym_{i,A}$ whenever desired.

*Pseudonym Conversion.* The CONVERT interface allows a server $\mathcal{S}_A$ to request the conversion of a previously obtained pseudonym $nym_{i,A}$ towards another server $\mathcal{S}_B$. Each conversion query gets associated with a unique identifier $cqid$. Upon each conversion request, the functionality internally creates a conversion record for $cqid$ that makes all conversions auditable for the underlying user. Recall that $\mathcal{F}$ knows the correlation between $nym_{i,A}$ and $\mathcal{U}_i$ and therefore can tailor the log entry to that user, without anyone else learning $\mathcal{U}_i$'s identity.

The converter $\mathcal{X}$ is then notified about the request, but only learns that $\mathcal{S}_A$ wants to run a conversion towards $\mathcal{S}_B$. If $\mathcal{X}$ or $\mathcal{S}_B$ are honest, the adversary does not learn anything about the pseudonym $nym_{i,A}$ the request was initiated for, and cannot even tell whether two requests are for the same pseudonym or not. If both $\mathcal{X}$ and $\mathcal{S}_B$ are corrupt, the adversary learns some partial information of $\mathcal{U}_i$ behind the pseudonym via the leakage function $\mathsf{leak}$.

When $\mathcal{X}$ allows the conversion, it completes the request via the PROCEED interface. The converted pseudonym $nym_{i,B}$ is then retrieved or generated using $\mathcal{F}$'s knowledge of the underlying $\mathcal{U}_i$ of $nym_{i,A}$. Further, the audit log entry gets updated to reflect completion of the conversion and the server $\mathcal{S}_B$ (and only $\mathcal{S}_B$) receives the converted pseudonym $nym_{i,B}$.

*User Audits.* The AUDIT interface allows a user $\mathcal{U}_i$ to retrieve a log file containing all conversion requests of his pseudonyms. This is done via the internal knowledge of $\mathcal{F}$ that stored all requests $cqid_j = (cqid'_j, \mathcal{S}_A, \mathcal{S}_B)$ associated with the affected user id $\mathcal{U}_i$. The interface only returns the entries stored for the requesting $\mathcal{U}_i$, thereby naturally enforcing that the individual conversion records are only accessible to the legitimate user.

*Leakage.* We parametrize our functionality with a leakage function leak, and hand $\ell \leftarrow \mathsf{leak}(\mathcal{U}_i)$ to the adversary whenever a pseudonym is generated or converted via a corrupt converter towards a corrupt server. Thus, the guaranteed privacy properties of $\mathcal{F}$ in the presence of a corrupt $\mathcal{X}$ depend on the function leak.

Our protocol in Section 4 realizes $\mathcal{F}$ with leak being a deterministic one-way function. That is, the leakage $\ell \leftarrow \mathsf{leak}(\mathcal{U}_i)$ still hides the user identity from the adversary but makes related pseudonyms linkable among corrupt servers, as $\ell$ is unique for each $\mathcal{U}_i$. Note that the leakage never allows the adversary to link pseudonyms between corrupt and honest servers, as the functionality enforces $\ell \leftarrow \perp$ whenever an honest server is involved.

Leaking the relation between pseudonyms owned by corrupt servers is in fact unavoidable in the presence of a corrupt converter: The corrupt converter and corrupt servers can internally link all the related pseudonyms they have. In the CL-15 functionality this was modeled more implicitly, as the adversary could simply use the CONVERT interface to convert any learned pseudonym of a corrupt server towards any other corrupt server (if the converter is corrupt). Here, however, this would not be possible as the functionality creates audit entries for the user upon each conversion. That is, the adversary can no longer use the CONVERT interface to link pseudonyms "off-the-record". Thus, we had to model this capability in a dedicated way without triggering audit log entries, which is exactly what our leakage handle $\ell$ does. Parametrizing $\mathcal{F}$ with the leakage function further makes the definition more flexible, as simpler protocols may exist that come for the price of increased leakage.

## 3. Building Blocks

In this section we introduce the building blocks required by our protocol. We start with the standard primitives of NIZK proofs, CPA-secure encryption and PRFs and then introduce our new building blocks of encryption with randomizable keys, oblivious pseudorandom function with committed outputs and signatures on encoded messages.

### 3.1. Bilinear Maps

Let $\mathbb{G}$, $\tilde{\mathbb{G}}$, and $\mathbb{G}_t$ be groups of prime order $q$. A map $e : \mathbb{G} \times \tilde{\mathbb{G}} \rightarrow \mathbb{G}_t$ must satisfy 1) bilinearity, i.e., $e(g^x, \tilde{g}^y) = e(g, \tilde{g})^{xy}$; 2) non-degeneracy, i.e., for all generators $g \in \mathbb{G}$ and $\tilde{g} \in \tilde{\mathbb{G}}$, $e(g, \tilde{g})$ generates $\mathbb{G}_t$; and 3) efficiency, i.e., there exists an efficient algorithm $\mathcal{G}(1^\tau)$ that outputs the bilinear group $(q, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_t, e, g, \tilde{g})$ and an efficient algorithm to compute $e(a, b)$ for any $a \in \mathbb{G}$ and $b \in \tilde{\mathbb{G}}$. If $\mathbb{G} = \tilde{\mathbb{G}}$ the map is symmetric and otherwise asymmetric.

### 3.2. Non-Interactive Proofs of Knowledge

By $\mathit{NIZK}\{(w) : \mathit{statement}(w)\}$ we denote a generic non-interactive zero-knowledge proof protocol of knowledge of a witness $w$ such that the $\mathit{statement}(w)$ is true. Sometimes we need witnesses to be online-extractable, which we make explicit by denoting with $\mathit{NIZK}\{(\underline{w_1}, w_2) : \mathit{statement}(w_1, w_2)\}$ the proof of witnesses $w_1$ and $w_2$, where $w_1$ can be extracted. For the ease of presentation we sometimes use simplified statements, e.g., we write $C = \mathsf{Enc}(epk, m)$ to describe a proof that $C$ is a proper encryption of $m$ under key $epk$, and omit the explicit randomness used within Enc. We use NIZKs for notational convenience, in our scheme all proofs could also be done interactively.

For concrete realizations of NIZKs, i.e., generalized Schnorr-signature proofs [6], we will use notation [9] such as $\mathit{SPK}\{(a, b, c) : y = g^a h^b \wedge \tilde{y} = \tilde{g}^a \tilde{h}^c\}$. Notice that, because the function $e(\cdot, g)$ is a group homomorphism, $\mathit{SPK}\{(a) : y = e(a, g)\}$ is a valid proof specification.

### 3.3. Commitment Scheme

We require a commitment scheme $(\mathsf{Com}_{\mathbb{G}}, \mathsf{ComVf}_G)$ for elements of $\mathbb{G}$. The algorithm $(com, o) \xleftarrow{\$} \mathsf{Com}_{\mathbb{G}}(m)$ on input a message $m \in \mathbb{G}$, produces a commitment $com$ and so-called *opening information* $o$. The commitment verification algorithm $\{0, 1\} \leftarrow \mathsf{ComVf}_{\mathbb{G}}(m, com, o)$ verifies whether $m$ is indeed committed to in a commitment $com$.

A suitable commitment scheme can be instantiated as an ElGamal encryption of $m$ under a random public key for which the corresponding secret key is not known. To this end, assume that in addition to $g$ a second generator $h \in \mathbb{G}$ is available, s.t. the discrete logarithm between $g$ and $h$ is unknown. Then, the algorithm $\mathsf{Com}_{\mathbb{G}}(m)$ first chooses a random $o \leftarrow \mathbb{Z}_q$ and computes and outputs $com \leftarrow (h^o, g^o \cdot m)$. The $\mathsf{ComVf}_{\mathbb{G}}(m, com, o)$ algorithms outputs 1 if $com = (h^o, g^o \cdot m)$ and 0 otherwise. Clearly, this commitment scheme is unconditionally binding ($m$ is uniquely defined by the commitment) and computationally hiding (under the Decisional Diffie-Hellman assumption).

### 3.4. Homomorphic Encryption Schemes

We require an encryption scheme $(\mathsf{EncKGen}_{\mathbb{G}}, \mathsf{Enc}_{\mathbb{G}}, \mathsf{Dec}_{\mathbb{G}})$ that is chosen-plaintext (CPA) secure and that has a cyclic group $\mathbb{G}$ as message space. It consists of a key generation algorithm $(epk, esk) \xleftarrow{\$} \mathsf{EncKGen}_{\mathbb{G}}(1^{\tau})$, where $\tau$ is a security parameter, an encryption algorithm $C \xleftarrow{\$} \mathsf{Enc}_{\mathbb{G}}(epk, m)$, with $m \in \mathbb{G}$, and a decryption algorithm $m \leftarrow \mathsf{Dec}_{\mathbb{G}}(esk, C)$. Sometimes we will make the randomness used in the encryption process explicit, in which case we will write $C \leftarrow \mathsf{Enc}_{\mathbb{G}}(epk, m, r)$, where $r$ encodes all the randomness, i.e., $\mathsf{Enc}_{\mathbb{G}}(\cdot, \cdot, \cdot)$ is a deterministic algorithm.

We require further that the encryption scheme has an appropriate *homomorphic property*, namely that there is an efficient operation $\odot$ on ciphertexts such that, if $C_1 \in \mathsf{Enc}_{\mathbb{G}}(epk, m_1)$ and $C_2 \in \mathsf{Enc}_{\mathbb{G}}(epk, m_2)$, then $C_1 \odot C_2 \in \mathsf{Enc}_{\mathbb{G}}(epk, m_1 \cdot m_2)$. We use exponentiation to denote the repeated application of $\odot$, e.g., $C^3$ denotes $C \odot C \odot C$.

*ElGamal Encryption (with a CRS Trapdoor).* We use the ElGamal encryption scheme, which is homomorphic and chosen plaintext secure. The CPA security is sufficient for our construction, as the parties always prove to each other that they formed the ciphertexts correctly. Let $(\mathbb{G}, g, q)$ be system parameters available as CRS such that the DDH problem is hard w.r.t. $\tau$.

$\mathsf{EncKGen}_{\mathbb{G}}(1^{\tau})$ : Pick random $x$ from $\mathbb{Z}_q$, compute $y \leftarrow g^x$, and output $esk \leftarrow x$ and $epk \leftarrow y$.

$\mathsf{Enc}_{\mathbb{G}}(epk, m)$ : To encrypt a message $m \in \mathbb{G}$ under $epk = y$, pick $r \xleftarrow{\$} \mathbb{Z}_q$ and output the ciphertext $(C_1, C_2) \leftarrow (y^r, g^r m)$.

$\mathsf{Dec}_{\mathbb{G}}(esk, C)$ : On input a secret key $esk = x$ and ciphertext $C = (C_1, C_2) \in \mathbb{G}^2$, output $m' \leftarrow C_2 \cdot C_1^{-1/x}$.

In our concrete instantiation we will use a variation of ElGamal encryption with a CRS trapdoor, which allows to make proofs for correct ciphertexts efficiently online extractable. That is, we assume that the CRS additionally contains a public key $\hat{y}$ and extend each ciphertext with an element $C_0 \leftarrow \hat{y}^r$, that is ignored in normal decryption. In the security proof of our protocol, the simulator will be privy to $\hat{x} = \log_g \hat{y}$ as it can set the CRS appropriately and thus is able to decrypt as $m' \leftarrow C_2 \cdot C_0^{-1/\hat{x}}$.

## 3.5. Re-randomizable Public Keys

In our construction we will need a second encryption scheme that, on top of being CPA-secure and homomorphic, allows re-randomization of the public keys. We call such a scheme a key-randomizable encryption scheme $(\mathsf{REncKGen}_{\mathbb{G}}, \mathsf{REnc}_{\mathbb{G}}, \mathsf{RDec}_{\mathbb{G}}, \mathsf{Rand}_{\mathbb{G}})$. Besides the standard algorithms to encrypt and decrypt, it also comes with a key-randomization algorithm $\mathsf{Rand}_{\mathbb{G}}$ that on input a public key $epk$ outputs a re-randomized key $epk'$. For our construction, the scheme must also allow for efficient proofs of correct re-randomization: $\pi_{\mathsf{Rand}} \xleftarrow{\$} \mathrm{NIZK}\{(epk) : epk' = \mathsf{Rand}_{\mathbb{G}}(epk)\}$ and the public keys be group elements, i.e., $epk \in \mathbb{G}^n$ for some $n$. These requirements are easy to meet as we shall see. For completeness we require that decryption succeeds for all ciphertexts that are generated under correctly re-randomized keys.

*Key-Indistinguishability.* The security guarantee we need from a key-randomizable encryption scheme is that by seeing different re-randomized keys, an adversary cannot tell which of the keys belong together. The formal definition of this key-indistinguishability is given below.

***Definition 3.1.*** We say a key-randomizable encryption scheme is *key-indistinguishable* if for any efficient algorithm $\mathcal{A}$ the probability that the experiment given in Figure 2 returns 1 with probability significantly larger than $1/2$ is negligible (as a function of $\tau$).

```
**Experiment** $\mathsf{Exp}^{\mathsf{REnc\text{-}keyind}}_{\mathcal{A},\mathsf{REnc}_{\mathbb{G}}}(\mathbb{G}, \tau)$:
  $(epk_0, esk_0) \xleftarrow{\$} \mathsf{REncKGen}_{\mathbb{G}}(1^{\tau})$
  $(epk_1, esk_1) \xleftarrow{\$} \mathsf{REncKGen}_{\mathbb{G}}(1^{\tau})$
  $b \xleftarrow{\$} \{0, 1\}$
  $epk^* \xleftarrow{\$} \mathsf{Rand}_{\mathbb{G}}(epk_b)$
  $b' \xleftarrow{\$} \mathcal{A}(epk_0, epk_1, epk^*)$
  return 1 if $b' = b$
```

Figure 2: Key-Indistinguishability Experiment.

A similar property, called incomparable keys, was introduced by Waters et al. [24]. They consider derivation of several incomparable public keys from the *secret* key, though, instead of re-randomization based solely on the public key. Further, their security notion for incomparable keys gives the adversary access to a decryption oracle, which in one world contains two secret keys and in the other world only one. Thus, the notion can only be satisfied for schemes that are also *robust* [1], i.e., where it is infeasible to come up with ciphertexts that decrypt under more than one secret key.

In a recent paper by Young and Yung [27], the notion of incomparable keys is adapted to re-randomization of public keys and the security definition does not enforce robustness of ciphertexts anymore (as they consider incomparable keys in isolation, without talking about encryption). However, their security notion is given for a specific ElGamal-based construction instead for the general class of schemes. Our definition above is a generalization of that notion.

*ElGamal-based Instantiation.* For our instantiation we use ElGamal encryption, which can easily be modified to become a secure key-randomizable encryption scheme. The randomization technique for the public key was also used by Waters et al. [24] and Young and Yung [27]. However in [24] the enc/decryption algorithms differ due to the additional robustness requirement, and [27] omits them as key-randomization was considered in isolation. But its easy to see how ElGamal encryption and decryption need to be adapted to cope with such randomizable keys. We call the following scheme R-ElGamal.

$\mathsf{REncKGen}_{\mathbb{G}}(1^{\tau})$ : Pick random $x \xleftarrow{\$} \mathbb{Z}_q$, $r \xleftarrow{\$} \mathbb{Z}_q$, compute $y_1 \leftarrow g^{xr}, y_2 \leftarrow g^r$, and output $esk \leftarrow x$ and $epk \leftarrow (y_1, y_2) \in \mathbb{G}^2$.

$\mathsf{Rand}_{\mathbb{G}}(epk)$ : On input a public key $epk = (y_1, y_2) \in \mathbb{G}^2$ choose a random $r' \xleftarrow{\$} \mathbb{Z}_q$ and output $epk' \leftarrow (y_1^{r'}, y_2^{r'}) \in \mathbb{G}^2$.

$\mathsf{REnc}_{\mathbb{G}}(epk, m)$ : To encrypt a message $m \in \mathbb{G}$ under $epk = (y_1, y_2)$, choose a random $r \xleftarrow{\$} \mathbb{Z}_q$ and output the ciphertext $(C_1, C_2) \leftarrow (y_1^r, y_2^r m)$.

$\mathsf{RDec}_{\mathbb{G}}(esk, C)$ : On input a secret key $esk = x$, ciphertext $C = (C_1, C_2) \in \mathbb{G}^2$, output $m' \leftarrow C_2 \cdot C_1^{-1/x}$.

***Theorem 3.2.*** (SECURITY OF R-ELGAMAL.) The key-randomizable encryption scheme $(\mathsf{REncKGen}_{\mathbb{G}}, \mathsf{Rand}_{\mathbb{G}}, \mathsf{REnc}_{\mathbb{G}}, \mathsf{RDec}_{\mathbb{G}})$ defined above is key-indistinguishable and IND-CPA secure under the DDH assumption.

Young and Yung proved $\mathsf{Rand}_{\mathbb{G}}$ to produce indistinguishable public keys based on the DDH assumption, deploying a generalization of the random self-reduction of DDH. Using the same arguments, the proof of key-indistinguishable is straightforward and omitted here. Similarly, it is easy to see that R-ElGamal is still IND-CPA secure.

## 3.6. Pseudorandom Functions

For the generation of pseudonyms we require a pseudorandom function, consisting of a key generation $k \xleftarrow{\$} \mathsf{PRFKGen}_{\mathbb{G}}(1^\tau)$ and evaluation function $z \leftarrow \mathsf{PRF}_{\mathbb{G}}(k, m)$. Apart from the standard pseudorandomness property, we also need $\mathsf{PRF}_{\mathbb{G}}$ to be a one-way function. That is, even when privy of the key, the function should be hard to invert on random inputs. We further need the function $\mathsf{PRF}_{\mathbb{G}}$ to be amendable to an oblivious evaluation protocol, which we will discuss later.

The construction by Dodis and Yampolskiy [12] satisfies all these requirements. Their $\mathsf{PRF}_{\mathbb{G}}$ works in a cyclic group $\mathbb{G} = \langle g \rangle$ of order $q$ and computes $\mathsf{PRF}_{\mathbb{G}}(k, m) = g^{1/(k+m)}$ for keys $k \xleftarrow{\$} \mathbb{Z}_q$. Pseudorandomness is based on the $q$-Decisional Diffie-Hellman Inversion problem [5] and, as is not hard to see, so is one-wayness.

We also need a pseudorandom *permutation*. It consists of a key generation $k \xleftarrow{\$} \mathsf{PRPKGen}_{\mathbb{G}}(1^\tau)$, a function $z \leftarrow \mathsf{PRP}_{\mathbb{G}}(k, m)$ and its efficiently computable inverse $m \leftarrow \mathsf{PRP}_{\mathbb{G}}^{-1}(k, z)$. For simplicity, we assume $\mathsf{PRP}_{\mathbb{G}}$ to work in a group $\mathbb{G}$ as well.

## 3.7. Multi-Session Oblivious Pseudorandom Function with Committed Output

Because users' pseudonyms need to be generated blindly by the converter, we use an Oblivious Pseudorandom Function (OPRF). The primitive of OPRF was introduced by Jarecki and Liu [18] and later refined by Jarecki et al. [17]. To be useful to us, however, we need to extend this primitives in two ways. First, we require that multiple invocations of the OPRF will use the same seed (which the functionality by Jarecki and Liu does not enforce). To this end, we define a multi session functionality where the sender provides the functionality with a seed upon setup. Later, receivers can call the functionality with a message $m$ and will get as a result the PRF applied to this message, where the same seed is used for all messages. Second, our functionality does provide the sender with a commitment to the receiver's output. This allows one to use the OPRF as modular building block, as the receiver can now prove to the sender that he used the correct results of the OPRF in some later computations.

Our functionality $\mathcal{F}_{\mathsf{cOPRF}}$ incorporating these two extensions is given in Figure 3. It is parametrized with $\mathsf{PRF}_{\mathbb{G}}$ and a commitment scheme $\mathsf{Com}_{\mathbb{G}}$.

1) `Setup`. On input of $(\mathsf{SETUP}, sid)$ from sender $\mathcal{S}$:
   - Draw $k \xleftarrow{\$} \mathsf{PRFKGen}_{\mathbb{G}}(1^\tau)$ and store record $(sid, k)$.
   - If $\mathcal{S}$ is corrupt, send output $(\mathsf{SETUP}, sid, k)$ to $\mathcal{S}$, otherwise, send public delayed output $(\mathsf{SETUP}, sid)$ to $\mathcal{S}$.
2) `Evaluation`. On input of $(\mathsf{PRF}, sid, qid, m)$ from $\mathcal{R}_i$:
   - Abort if record $(qid, \mathcal{R}_i, \cdot)$ exists or no $(sid, k)$ is stored.
   - Compute $z \leftarrow \mathsf{PRF}_{\mathbb{G}}(k, m)$.
   - Store record $(qid, \mathcal{R}_i, z)$.
   - Send delayed private output $(\mathsf{PRF}, sid, qid, z)$ to $\mathcal{R}_i$.
3) `Commit`. On input of $(\mathsf{COM}, sid, qid, com_z, o_z)$ from $\mathcal{R}_i$:
   - Proceed only if a record $(qid, \mathcal{R}_i, z)$ for $qid$ and $\mathcal{R}_i$ exists s.t. $1 = \mathsf{ComVf}_{\mathbb{G}}(z, com_z, o_z)$.
   - Send public delayed output $(\mathsf{COM}, sid, qid, com_z)$ to $\mathcal{S}$.

Figure 3: Ideal functionality $\mathcal{F}_{\mathsf{cOPRF}}$ with $sid = (sid', \mathcal{S})$ .

*$\mathcal{F}_{\mathsf{cOPRF}}$ Realization.* We now provide a concrete realization $\Pi_{\mathsf{cOPRF}}$ of this functionality for the Dodis-Yampolskiy PRF. Our protocol $\Pi_{\mathsf{cOPRF}}$ is inspired by the protocol that Belenkiy et al. [4] have designed to let an issuer blindly sign a random user secret key. The basic idea is to use the homomorphic properties in the exponents and deploy the semantically secure version of the Camenisch-Shoup encryption scheme [8] $(\mathsf{EncKGen}_n, \mathsf{Enc}_n, \mathsf{Dec}_n)$ to jointly compute $\mathsf{PRF}_{\mathbb{G}}(k, m) = g^{1/(k+m)}$. The protocol is given in Figure 4 and requires just a few exponentiations in $\mathbb{G}$ and in $\mathbb{Z}_{n^2}^*$ of both sender and receiver, where $n$ is the RSA modulus of the Camenisch-Shoup encryption scheme. The proof that $\Pi_{\mathsf{cOPRF}}$ securely realizes $\mathcal{F}_{\mathsf{cOPRF}}$ for honest-but-curious $\mathcal{S}$ is given in the full paper.

***Theorem 3.3.*** $\Pi_{\mathsf{cOPRF}}$ is a secure realization of $\mathcal{F}_{\mathsf{cOPRF}}$ in the $(\mathcal{F}_{\mathsf{1\text{-}AUTH}}, \mathcal{F}_{\mathsf{CA}}, \mathcal{F}_{\mathsf{CRS}})$-hybrid model for the Dodis-Yampolskiy PRF under the q-Decisional Diffie-Hellman Inversion (q-DBDHI) and the Decision Composite Residuosity (DCR) assumptions and if the sender is at most honest-but-curiously corrupted.

## 3.8. Homomorphic Encoding Functions

Let $\{(\mathsf{Ef}_{\mathbb{G}}^{(i)}, \mathsf{Df}_{\mathbb{G}}^{(i)})\}$ be a family of pairs of probabilistic encoding and decoding functions for a group $\mathbb{G}$, where $\mathsf{Ef}_{\mathbb{G}}^{(i)}$ is a probabilistic function $\mathsf{Ef}_{\mathbb{G}}^{(i)} : \mathbb{G} \to \{0,1\}^*$, $\mathsf{Df}_{\mathbb{G}}^{(i)}$ a function $\mathsf{Df}_{\mathbb{G}}^{(i)} : \{0,1\}^* \to \mathbb{G}$, and for all $m \in \mathbb{G}$ we have that $m = \mathsf{Df}_{\mathbb{G}}^{(i)}(\mathsf{Ef}_{\mathbb{G}}^{(i)}(m))$ holds. We require the encoding functions to be homomorphic, namely that there is an efficient operation $\odot$ that, for all $C_1 \in \mathsf{Ef}_{\mathbb{G}}^{(i)}(m_1)$ and

1) `Setup`. On input of $(\mathsf{Setup}, sid)$ with $sid = (sid', \mathcal{S})$ sender $\mathcal{S}$ executes:

- Choose $k \xleftarrow{\$} \mathbb{Z}_q$, generate $(epk, spk) \xleftarrow{\$} \mathsf{EncKGen}_n(1^\tau)$, and compute $C_k \xleftarrow{\$} \mathsf{Enc}_n(epk, k)$.
- Store $(sid, k, epk, esk)$ and register $(epk, C_k)$ with $\mathcal{F}_{\mathsf{CA}}$ for session-id $(sid', oprf, \mathcal{S})$.
- Output $(\mathsf{SETUP}, sid)$.

2) `Evaluation`. On input of $(\mathsf{EVAL}, sid, qid, m)$ to receiver $\mathcal{R}_i$, the following protocol between $\mathcal{R}_i$ and $\mathcal{S}$ is executed.

- $\mathcal{R}_i$, when executed for the first time, parses $sid$ as $(sid', \mathcal{S})$ and retrieves $(epk, C_k)$ from $\mathcal{F}_{\mathsf{CA}}$ with session id $(sid', oprf, \mathcal{S})$, and stores these parameters together with $(sid, epk, C_k)$. Otherwise, $\mathcal{R}_i$ retrieves $(sid, epk, C_k)$ from storage.
- $\mathcal{R}_i$ computes encrypted input to the PRF:
  - Choose $r \xleftarrow{\$} \mathbb{Z}_q$ and compute $C_v \xleftarrow{\$} (C_k \cdot \mathsf{Enc}_n(epk, m))^r$, $(com_r, o_r) \xleftarrow{\$} \mathsf{Com}_{\mathbb{G}}(g^r)$, and prove correctness of these values in $\pi_1 \xleftarrow{\$} \mathrm{NIZK}\{(m, \underline{r}, o_r) : (com_r, o_r) = \mathsf{Com}_{\mathbb{G}}(g^r) \ \wedge \ (C_k \cdot \mathsf{Enc}_n(epk, m))^r\}$.
  - Send $(\mathsf{EVAL}_1, sid, qid, com_r, C_v, \pi_1)$ to $\mathcal{S}$ via $\mathcal{F}_{\mathsf{1\text{-}AUTH}}$.
- $\mathcal{S}$ upon receiving $(\mathsf{EVAL}_1, sid, qid, com_r, C_v, \pi_1)$ blindly computes $g^{1/r(k+m)}$ on the encrypted input:
  - Verify $\pi_1$ (if it fails, send $\mathcal{R}_i$ an abort message) and compute $v \leftarrow \mathsf{Dec}_n(esk, C_v)$ and $V \leftarrow g^{1/v}$.
  - Store $(sid, qid, V, com_r)$ and send $(\mathsf{EVAL}_2, sid, qid, V)$ to $\mathcal{R}_i$ via $\mathcal{F}_{\mathsf{1\text{-}AUTH}}$.
- $\mathcal{R}_i$ upon receiving $(\mathsf{EVAL}_2, sid, qid, V)$ unblinds the received value:
  - Compute $z \leftarrow V^r$, store $(qid, V, r, com_r, o_r, z)$, and output $(\mathsf{PRF}, sid, qid, z)$.

3) `Commit`. On input of $(\mathsf{COM}, sid, qid, com_z, o_z)$ to $\mathcal{R}_i$, the following protocol is executed.

- $\mathcal{R}_i$ proves that $com_z$ is a correct commitment on $z = V^r$:
  - Retrieve record $(qid, V, r, com_r, o_r, z)$ for $qid$, check that $1 = \mathsf{ComVf}_{\mathbb{G}}(z, com_z, o_z)$, and prove correctness of the commitment in $\pi_2 \xleftarrow{\$} \mathrm{NIZK}\{(z, \underline{o_z}, r, o_r) : (com_r, o_r) = \mathsf{Com}_{\mathbb{G}}(g^r) \ \wedge \ (com_z, o_z) = \mathsf{Com}_{\mathbb{G}}(V^r)\}$.
  - Send $(\mathsf{COM}_1, sid, qid, (com_z, \pi_2))$ to $\mathcal{S}$ via $\mathcal{F}_{\mathsf{1\text{-}AUTH}}$.
- $\mathcal{S}$ upon input $(\mathsf{COM}_1, sid, qid, (com_z, \pi_2))$ verifies $\pi_2$ w.r.t. $V$ and $com_r$ stored for $qid$ and outputs $(\mathsf{COM}, sid, qid, com_z)$.

Figure 4: Realization $\Pi_{\mathsf{cOPRF}}$ of $\mathcal{F}_{\mathsf{cOPRF}}$ for an honest-but-curious sender. The realization is also parametrized with a commitment scheme $(\mathsf{Com}_{\mathbb{G}}, \mathsf{ComVf}_{\mathbb{G}})$. All communication between the sender and the receiver are via the one-sided authenticated channel functionality $\mathcal{F}_{\mathsf{1\text{-}AUTH}}$, where $\mathcal{S}$ is the authenticated party. Unless otherwise specified, if a check fails, the checking party aborts.

$C_2 \in \mathsf{Ef}^{(i)}_{\mathbb{G}}(m_2)$, then $C_1 \odot C_2 \in \mathsf{Ef}^{(i)}_{\mathbb{G}}(m_1 \cdot m_2)$. We again use exponentiation to denote the repeated application of $\odot$.

Let us consider some examples of encoding functions. Let $(\mathsf{EncKGen}_{\mathbb{G}}, \mathsf{Enc}_{\mathbb{G}}, \mathsf{Dec}_{\mathbb{G}})$ be a homomorphic semantically secure encryption scheme and let $(epk_{(i,j)}, esk_{(i,j)})$ be key pairs for it. Then the two pairs of functions given below are members of the family $\{(\mathsf{Ef}^{(i)}_{\mathbb{G}^3}, \mathsf{Df}^{(i)}_{\mathbb{G}^3})\}$, where the operators '$\cdot$' and '$\odot$' are defined component wise.

$$\mathsf{Ef}^{(1)}_{\mathbb{G}^3}(m_1, \ldots, m_3) = (m_1, \mathsf{Enc}_{\mathbb{G}}(epk_{(1,2)}, m_2), \mathsf{Enc}_{\mathbb{G}}(epk_{(1,3)}, m_3))$$

$$\mathsf{Df}^{(1)}_{\mathbb{G}^3}(c_1, \ldots, c_3) = (c_1, \mathsf{Dec}_{\mathbb{G}}(esk_{(1,2)}, c_2), \mathsf{Dec}_{\mathbb{G}}(esk_{(1,3)}, c_3))$$

## 3.9. A Signature Scheme for Homomorphic Message Encoding Functions

We require a signature scheme that is compatible with a family $\{(\mathsf{Ef}^{(i)}_{\mathbb{G}}, \mathsf{Df}^{(i)}_{\mathbb{G}})\}$ of homomorphic message encoding functions, i.e., the scheme is able to sign messages that are encoded with a function $\mathsf{Ef}_{\mathbb{G}}$ where for some $\mathsf{Df}_{\mathbb{G}}$ the pair $(\mathsf{Ef}_{\mathbb{G}}, \mathsf{Df}_{\mathbb{G}})$ is a member of the family $\{(\mathsf{Ef}^{(i)}_{\mathbb{G}}, \mathsf{Df}^{(i)}_{\mathbb{G}})\}$.

Such a signature scheme is a generalization of the dual-mode signature scheme as defined by Camenisch and Lehmann [7]. While their scheme has two signing algorithms (i.e., two modes), in one mode messages are signed in the clear and in the other mode encrypted messages are signed, we unify these two signing algorithms into a single one. Furthermore, we generalize the signature scheme so that blocks of messages can be signed (some of the messages in the clear, some of them encrypted).

A signature scheme for a family of homomorphic message encoding functions with message space $\mathbb{G}$ consists of four algorithms $(\mathsf{SigKGen}_{\mathbb{G}}, \mathsf{EncSign}_{\mathbb{G}}, \mathsf{DecSign}_{\mathbb{G}}, \mathsf{Vf}_{\mathbb{G}})$.

$\mathsf{SigKGen}_{\mathbb{G}}(1^\tau)$ : On input the security parameter and being parametrized by $\mathbb{G}$, this algorithm outputs a public verification key $spk$ and secret signing key $ssk$.

$\mathsf{EncSign}_{\mathbb{G}}(ssk, \mathsf{Ef}_{\mathbb{G}}, C)$ : On input a signing key $ssk$, encoding function $\mathsf{Ef}_{\mathbb{G}}$, and an encoding $C$, the signing algorithm outputs an "encoded" signature $\overline{\sigma}$ of $C$.

$\mathsf{DecSign}_{\mathbb{G}}(spk, \mathsf{Df}_{\mathbb{G}}, \overline{\sigma})$ : On input an "encoded" signature $\overline{\sigma}$, decoding function $\mathsf{Df}_{\mathbb{G}}$, and public key $spk$, this algorithm outputs a "decoded" signature $\sigma$.

$\mathsf{Vf}_{\mathbb{G}}(spk, \sigma, m)$ : On input a public verification key $spk$, signature $\sigma$, and message $m \in \mathbb{G}$, this algorithm outputs 1 if the signature is valid and 0 otherwise.

Compatibility of the signature scheme with the homomorphic encoding now means that signatures $\overline{\sigma} \xleftarrow{\$} \mathsf{EncSign}_{\mathbb{G}}(ssk, \mathsf{Ef}_{\mathbb{G}}, C)$ obtained on an encoding $C \xleftarrow{\$} \mathsf{Ef}_{\mathbb{G}}(m)$, can be decoded to a valid signature $\sigma \leftarrow \mathsf{DecSign}_{\mathbb{G}}(spk, \mathsf{Df}_{\mathbb{G}}, \overline{\sigma})$ on $m$.

*Security Definition.* The security definition of a signature scheme for a family of homomorphic encoding function is close to that of unforgeability for an ordinary signature scheme, the main difference being that 1) we consider $m$ to be a vector of messages which 2) can be homomorphically encoded. For the latter, we assume for simplicity that the encoding is done correctly, either because the signer encoded the messages himself or the party providing the encoded message proves to the signer that the encoding was done correctly. Figure 5 provides the corresponding security experiment.

```
**Experiment** $\mathsf{Exp}^{\mathsf{ENCSIG\text{-}forge}}_{\mathcal{A},\mathsf{ENCSIG},\mathsf{Ef}_{\mathbb{G}}}(\tau, \mathbb{G}, \{(\mathsf{Ef}^{(i)}_{\mathbb{G}}, \mathsf{Df}^{(i)}_{\mathbb{G}})\})$:
  $(spk, ssk) \xleftarrow{\$} \mathsf{SigKGen}_{\mathbb{G}}(1^{\tau})$
  $\mathbf{L} \leftarrow \emptyset$
  $(m^*, \sigma^*) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{Sign}}(ssk,\cdot,\cdot)}(spk)$
    where $\mathcal{O}_{\mathsf{Sign}}$ on input $(\mathsf{Ef}^{(i)}_{\mathbb{G}}, m_i)$:
      adds $m_i$ to the list of queried messages $\mathbf{L} \leftarrow \mathbf{L} \cup m_i$
      runs $C_i \xleftarrow{\$} \mathsf{Ef}^{(i)}_{\mathbb{G}}(m_i)$
      computes $\overline{\sigma}_i \xleftarrow{\$} \mathsf{EncSign}_{\mathbb{G}}(ssk, \mathsf{Ef}^{(i)}_{\mathbb{G}}, C_i)$
      returns $(\overline{\sigma}_i, C_i)$
  return 1 if $\mathsf{Vf}_{\mathbb{G}}(spk, \sigma^*, m^*) = 1$ and $m^* \notin \mathbf{L}$
```

Figure 5: Unforgeability experiment for a signature scheme for a family of homomorphic encoding functions.

***Definition 3.4.*** We say that a signature scheme for encoded messages is *unforgeable* against adaptively chosen message attacks if for any efficient algorithm $\mathcal{A}$ the probability that the experiment in Figure 5 returns 1 is negligible (in the security parameter $\tau$).

*Instantiation.* We now give an instantiation of a signature scheme for any family $\{(\mathsf{Ef}^{(i)}_{\mathbb{G}^n}, \mathsf{Df}^{(i)}_{\mathbb{G}^n})\}$ of encoding functions for product groups $\mathbb{G}^n$ for some constant $n$. This means $\mathbb{G}^n$ needs to be the message space of the signature scheme. To this end, we extend a recent structure-preserving signature scheme by Groth [15] that works in a bilinear maps setting. We denote this the Gr signature scheme. That scheme is defined to sign a matrix of group elements, here we consider the special case where we sign only a vector of $n$ group elements. We recall this special case of the Gr scheme $(\mathsf{SigKGen}_{\mathbb{G}^n}, \mathsf{Sign}_{\mathbb{G}^n}, \mathsf{Vf}_{\mathbb{G}^n})$, slightly adapted to our notation, and then describe how to instantiate the additional algorithms $\mathsf{EncSign}_{\mathbb{G}^n}$ and $\mathsf{DecSign}_{\mathbb{G}^n}$.

The signature scheme assumes the availability of system parameters $crs = (q, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_t, e, g, \tilde{g}, x_1, \dots, x_n)$ consisting of $(q, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_t, e, g, \tilde{g}) \xleftarrow{\$} \mathcal{G}(1^{\tau})$ and $n$ additional random group elements $x_i \xleftarrow{\$} \mathbb{G}$.

$\mathsf{SigKGen}_{\mathbb{G}^n}(q, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_t, e, g, \tilde{g}, x_1, \dots, x_n)$ : Choose $v \xleftarrow{\$} \mathbb{Z}_q$, compute $y \leftarrow \tilde{g}^v$, and return $spk = y$, $ssk = v$.

$\mathsf{Sign}_{\mathbb{G}^n}(ssk, (m_1, \dots, m_n))$ : On input a message $(m_1, \dots, m_n) \in \mathbb{G}^n$ and key $ssk = v$, choose a random $u \xleftarrow{\$} \mathbb{Z}^*_q$, and output the signature $\sigma = (r, s, t_1, \dots, t_n)$, where $r \leftarrow \tilde{g}^u$, $s \leftarrow (x_1 \cdot g^v)^{1/u}$, and $t_i \leftarrow (m_i x_i^v)^{1/u}$.

$\mathsf{Vf}_{\mathbb{G}^n}(spk, \sigma, (m_1, \dots, m_n))$ : Parse input as $\sigma = (r, s, t_1, \dots, t_n)$ and $spk = y$ and accept if: $m_i, s, t_i \in \mathbb{G}$, $r \in \tilde{\mathbb{G}}$, $e(s, r) = e(g, y) \cdot e(x_1, \tilde{g})$, and $e(t_i, r) = e(m_i, \tilde{g}) \cdot e(x_i, y)$.

As pointed out by Groth, a signature $\sigma = (r, s, t_1, \dots, t_n)$ can be randomized to obtain a signature $\sigma' = (r', s', t'_1, \dots, t'_n)$ by picking a random $u' \xleftarrow{\$} \mathbb{Z}^*_q$ and computing $r' \leftarrow r^{u'}$, $s' \leftarrow s^{1/u'}$, and $t'_i \leftarrow t_i^{1/u'}$.

Now, we present the additional algorithms to extend the Gr signature scheme into one for the family of homomorphic message encoding functions $\{(\mathsf{Ef}^{(i)}_{\mathbb{G}^n}, \mathsf{Df}^{(i)}_{\mathbb{G}^n})\}$. We denote this scheme by Gr+. Let $(\mathsf{Ef}_{\mathbb{G}^n}, \mathsf{Df}_{\mathbb{G}^n})$ be an element of $\{(\mathsf{Ef}^{(i)}_{\mathbb{G}^n}, \mathsf{Df}^{(i)}_{\mathbb{G}^n})\}$ and $(C_1, \dots C_n) = \mathsf{Ef}_{\mathbb{G}^n}(m_1, \dots, m_n)$ be an encoding of the message $(m_1, \dots, m_n)$.

$\mathsf{EncSign}_{\mathbb{G}^n}(ssk, \mathsf{Ef}_{\mathbb{G}^n}, (C_1, \dots, C_n))$ : On input a secret key $ssk = v$ and a *correct* encoding $(C_1, \dots, C_n)$, choose a random $u \xleftarrow{\$} \mathbb{Z}^*_q$, and output the encoded signature $\overline{\sigma} = (r, s, T_1, \dots, T_n)$ with $r \leftarrow \tilde{g}^{1/u}$, $s \leftarrow (x_1 \cdot g^v)^u$, and $(T_1, \dots, T_n) \leftarrow ((C_1, \dots, C_n) \odot \mathsf{Ef}_{\mathbb{G}^n}(x_1^v, \dots, x_n^v))^u$.

$\mathsf{DecSign}_{\mathbb{G}^n}(spk, \mathsf{Df}_{\mathbb{G}^n}, \overline{\sigma})$ : Parse $\overline{\sigma} = (r, s, T_1, \dots, T_n)$, compute $(t_1, \dots, t_n) \leftarrow \mathsf{Df}_{\mathbb{G}^n}(T_1, \dots, T_n)$, and output $\sigma = (r, s, t_1, \dots, t_n)$.

It is not hard to see that $\sigma = (r, s, t_1, \dots, t_n)$ is a valid signature on the message $(m_1, \dots, m_n) \leftarrow \mathsf{Df}_{\mathbb{G}^n}(C_1, \dots, C_n)$, and that the distribution of the signature values is the same as when the messages $m_i$ were signed directly. The proof of the following theorem is provided in the full version.

***Theorem 3.5.*** If the Gr signature scheme $(\mathsf{SigKGen}_{\mathbb{G}}, \mathsf{Sign}_{\mathbb{G}}, \mathsf{Vf}_{\mathbb{G}})$ is unforgeable, then the scheme Gr+ resulting from adding the algorithms $\mathsf{EncSign}_{\mathbb{G}}$ and $\mathsf{DecSign}_{\mathbb{G}}$ described above, is an unforgeable signature scheme for $\{(\mathsf{Ef}^{(i)}_{\mathbb{G}^n}, \mathsf{Df}^{(i)}_{\mathbb{G}^n})\}$.

*Discussion.* Its algebraic properties allow one to integrate our signature scheme well into higher-level protocols, making it a very powerful building block. By construction it can be used to sign encrypted and committed messages. Furthermore, as verification of a signature consists of pairing equations only, one can do efficient proofs of knowledge of a signature on encrypted or committed messages with generalized Schnorr signature proofs. Also, it's not too hard to distribute the signing process and keys to multiple parties. Due to space constraints, we provide the details of this in the full version only and just mention that our new signature scheme is a very suitable basis to construct a variety of privacy-enabling schemes such as group signatures or anonymous credentials.

## 4. Our Protocol

The computation of pseudonyms in our protocol roughly follows the protocol by Camenisch and Lehmann [7], but adds audit capabilities for users and allows pseudonyms to be generated in a blind manner. We first briefly recall the pseudonym generation and conversion from their protocol (CL-15), and then discuss our extensions and changes.

Roughly, in the CL-15 protocol, a pseudonym consists of a random core $z_i$ that is unique per user, and to which the converter and server add server-specific randomness. The core is derived from the secret user identifier $uid_i$ as $z_i \leftarrow$

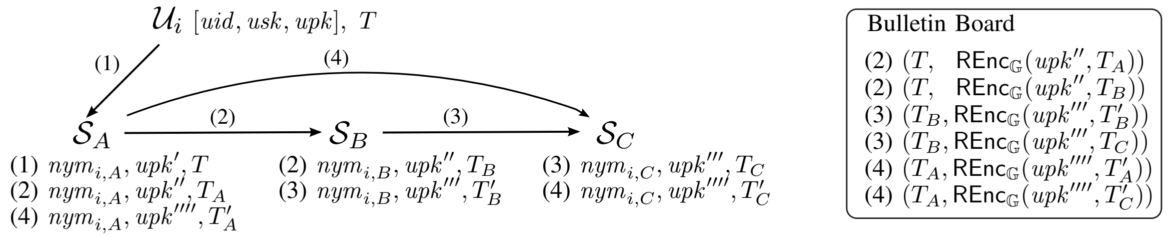

$\mathcal{U}_i$ $[uid, usk, upk]$, $T$
(1)
(4)
(2)
(3)
$\mathcal{S}_A$
$\mathcal{S}_B$
$\mathcal{S}_C$
(1) $nym_{i,A}, upk', T$
(2) $nym_{i,A}, upk'', T_A$
(4) $nym_{i,A}, upk'''', T'_A$
(2) $nym_{i,B}, upk'', T_B$
(3) $nym_{i,B}, upk''', T'_B$
(3) $nym_{i,C}, upk''', T_C$
(4) $nym_{i,C}, upk'''', T'_C$
Bulletin Board
(2) $(T,\ \mathsf{REnc}_{\mathbb{G}}(upk'', T_A))$
(2) $(T,\ \mathsf{REnc}_{\mathbb{G}}(upk'', T_B))$
(3) $(T_B, \mathsf{REnc}_{\mathbb{G}}(upk''', T'_B))$
(3) $(T_B, \mathsf{REnc}_{\mathbb{G}}(upk''', T_C))$
(4) $(T_A, \mathsf{REnc}_{\mathbb{G}}(upk'''', T'_A))$
(4) $(T_A, \mathsf{REnc}_{\mathbb{G}}(upk'''', T'_C))$

Figure 6: Example of the tag structure and encrypted tag-chain on the bulletin board after three conversions of $\mathcal{U}_i$'s pseudonyms. The tuples below the servers denote the pseudonym, public key and tag (for user $\mathcal{U}_i$) the servers hold after each conversion.

$\mathsf{PRF}_{\mathbb{G}}(k_{\mathcal{X}}, uid_i)$ where $k_{\mathcal{X}}$ is a secret key known only the converter. From the unique core identifier $z_i$, the converter then derives its pseudonym contribution $xnym_{i,A} \leftarrow z_i^{x_A}$ using a *secret* exponent $x_A$ that it chooses for each server $\mathcal{S}_A \in \mathbf{S}$, but never reveals to them.

To let the converter blindly convert a pseudonym from $\mathcal{S}_A$ to $\mathcal{S}_B$ the pseudonym is sent to $\mathcal{X}$ only in homomorphically encrypted form. That is, server $\mathcal{S}_A$ encrypts his pseudonym under the key of the second server $\mathcal{S}_B$ and sends the encryption to $\mathcal{X}$. The converter then uses the homomorphic properties of the encryption scheme and raises the encrypted pseudonym to the quotient of the two servers' secret keys, thereby blindly transforming the encrypted pseudonym.

*Blind Pseudonym Generation.* We no longer want the converter to learn the user identifier $uid_i$, nor the core identifier $z_i$ when generating a pseudonym from scratch for a user $\mathcal{U}_i$. We therefore let $\mathcal{U}_i$ engage in the pseudonym generation who then derives $z_i$ jointly with $\mathcal{X}$ using an oblivious pseudorandom function $\mathcal{F}_{\mathsf{cOPRF}}$. Thus, as in CL-15, the core is derived as $z_i \leftarrow \mathsf{PRF}_{\mathbb{G}}(k_{\mathcal{X}}, uid_i)$ but now without revealing $uid_i$ or $z_i$ to $\mathcal{X}$.

*User Audit.* The main challenge was to add efficient audit capabilities for the user without destroying or reducing the privacy guarantees of the pseudonym system. The core idea is to let the user also provide a public encryption key of a key pair $(upk, usk)$ when generating a pseudonym, and carry the public key along whenever the pseudonym gets converted. In a conversion request, the converter then receives an encrypted pseudonym and $upk$, such that $\mathcal{X}$ can encrypt some log information under the user's key and publish the ciphertext. Clearly, if $upk$ is a static key this would immediately destroy all our unlinkability and privacy features. We therefore use a key-randomizable encryption scheme that allows to obtain randomized public keys $upk' \xleftarrow{\$} \mathsf{Rand}_{\mathbb{G}}(upk)$ that are unlinkable to $upk$. In our protocol we then re-randomize the user's public key with every usage.

The converter could now simply encrypt the relevant audit information under $upk'$ and provide them on a public bulletin board. This would be secure but not very efficient: the user would have to decrypt all ciphertexts using $usk$ and for each check whether it decrypts to a valid audit log value. In a context such as a social security system with possibly billion of users, clearly, this would not yield a practical solution.

We therefore use a tag-chaining approach that makes the number of required decryptions independent of the amount of users. The main idea is to leverage the fact that in our protocol all pseudonyms originate from a user request. This allows us to give the user some initial "hook" – the audit tag – from which he can follow the propagation of his pseudonyms. That is, whenever a user $\mathcal{U}_i$ generates a pseudonym via the converter towards server $\mathcal{S}_A$, the pseudonym gets associated to a random audit tag $T$ that is known to the user and server, but not to the converter. Only when $\mathcal{S}_A$ wants to convert that pseudonym towards $\mathcal{S}_B$, it has to reveal $T$ to the converter who then publishes $T$ along with the conversion context encrypted under the user's randomized key. Since $T$ is known to the user, he only has to decrypt the audit entry for his tag.

However, we cannot allow $\mathcal{S}_A$ to re-use the same $T$ for another conversion of that pseudonym, as this would clearly break unlinkability of the conversion requests. Thus, in order to keep a used pseudonym "functional" for $\mathcal{S}_A$, he associates the pseudonym to a new random tag $T_A$ after every usage. To inform the (unknown) user behind the pseudonym about the new audit tag, $\mathcal{S}_A$ encrypts $T_A$ under $upk'$ and hands the ciphertext to $\mathcal{X}$ who publishes it along with the old tag $T$ at the bulletin board. Then, the user knowing $T$ and $usk$ – and *only* that user – can secretly obtain the new tag $T_A$ and follow the usage of the subsequent spending of his pseudonym by $\mathcal{S}_A$. Similarly, the receiving server $\mathcal{S}_B$ in the conversion request also chooses a fresh audit tag $T_B$ that he associates with the converted pseudonym. $\mathcal{S}_B$ encrypts the tag under $upk'$ and sends it to $\mathcal{X}$ where it gets published together with $T$. Again, only the user knowing $T$ and $usk$ can retrieve $T_B$ and follow the usage of his pseudonym with $\mathcal{S}_B$. An example of such a audit-tag chain is depicted in Figure 6.

*Security against Corrupted Parties.* We want our protocol to be tolerant to corrupt users and servers and a (honest-but-curious) corrupted converter, which requires some additions and modifications to the procedure described above. First, all audit tags must be generated jointly with the converter, that will blindly contribute randomness to each tag. This ensures uniqueness of tags even when triggered by corrupt users or servers, yet keeps the tag hidden from the converter during generation.

Further, we have to make sure that every occurrence of a pseudonym, randomized public key and tag is tightly bound together. That is, an attacker should not be able to reuse an honest users pseudonym out of context, e.g., along with an adversarially chosen public key, as this would break the privacy and auditability of our scheme. To prevent such attacks we use our signature scheme for blocks of encoded messages, which the converter uses to sign the tuple of pseudonym, public key and tag whenever generating or converting a pseudonym. The server then has to prove at every conversion request that it owns a signature on the correct set of values. The capability of signing *encoded* messages is crucial, as we will let $\mathcal{X}$ sign the tag and pseudonym only in encrypted form which ensures privacy even in presence of a corrupt converter.

## 4.1. Detailed Description

We now describe our protocol assuming that functionalities $\mathcal{F}_{\mathsf{CA}}, \mathcal{F}_{\mathsf{AUTH}}, \mathcal{F}_{\mathsf{1\text{-}AUTH}}, \mathcal{F}_{\mathsf{CRS}}, \mathcal{F}_{\mathsf{BB}}, \mathcal{F}_{\mathsf{cOPRF}}$ are available to all parties. The certificate authority functionality $\mathcal{F}_{\mathsf{CA}}$ [10] allows to register public keys, and we assume that parties call $\mathcal{F}_{\mathsf{CA}}$ to retrieve the necessary key material whenever they have to use a public key of another party. The authenticated channel functionality $\mathcal{F}_{\mathsf{AUTH}}$ [3] enables authenticated communication between two parties which we assume for all communication between servers and the converter. For communication between a user and the converter, we rely on a one-side authenticated channel $\mathcal{F}_{\mathsf{1\text{-}AUTH}}$ [3]. That is, only one side (in our protocol the converter) authenticates himself towards the other party (in our protocol the user). While the channel does not provide authenticity of the user, it still guarantees that all messages a converter receives in a given session originate from the same user. A natural realization of $\mathcal{F}_{\mathsf{1\text{-}AUTH}}$ is the TLS protocol with server certificates. The authenticated bulletin board $\mathcal{F}_{\mathsf{BB}}$ [25] allows a dedicated party to publish authenticated content that can be retrieved by any party. The common reference string functionality $\mathcal{F}_{\mathsf{CRS}}$ [10] provides all parties with the system parameters, consisting of the security parameter $\tau$ and a cyclic group $\mathbb{G} = \langle g \rangle$ of order $q$ (which is a $\tau$-bit prime), and the parameters for the NIZK proofs. And finally, $\mathcal{F}_{\mathsf{cOPRF}}$ allows to obliviously evaluate a pseudorandom function, as defined in Section 3.

To make the protocol more readable we often omit the explicit interface calls to those functionalities, and simply say that e.g., $\mathcal{S}_A$ sends a message to $\mathcal{X}$, or $\mathcal{X}$ publishes data at $\mathcal{F}_{\mathsf{BB}}$, instead of explicitly calling $\mathcal{F}_{\mathsf{AUTH}}$ or $\mathcal{F}_{\mathsf{BB}}$ with sub-session IDs etc.

**4.1.1. Setup.** To initiate our pseudonym system, the converter chooses a random string $sid'$ and sets the session identifier to be $sid = (sid', \mathcal{X}, \mathbf{S}, \mathbf{N})$ where $\mathbf{S} = \{\mathcal{S}_A, \mathcal{S}_B, \dots\}$ is the set of all server identities and $\mathbf{N} = \mathbb{G}$ defines the domain of the pseudonyms. The converter then makes $sid$ available to all participating entities, upon which they generate their respective keys as defined in Figure 7. Notice that we assume that user public keys are elements of $\mathbb{G}^n$, as already mentioned in the previous section. We further assume that all parties will retrieve and store the CRS via $\mathcal{F}_{\mathsf{CRS}}$ for $sid$.

```
Converter Setup:
  $(epk_{\mathcal{X}}, esk_{\mathcal{X}}) \xleftarrow{\$} \mathsf{EncKGen}_{\mathbb{G}}(1^\tau)$
  call $\mathcal{F}_{\mathsf{cOPRF}}$ with $(\mathsf{Setup}, (sid, \mathcal{X}))$
      (which internally creates $k_{\mathcal{X}} \xleftarrow{\$} \mathsf{PRFKGen}_{\mathbb{G}}(1^\tau)$)
  for each server $\mathcal{S}_A \in \mathbf{S}$:
      $(spk_{\mathcal{X},A}, ssk_{\mathcal{X},A}) \xleftarrow{\$} \mathsf{SigKGen}_{\mathbb{G}}(1^\tau)$
      choose a random $x_A \xleftarrow{\$} \mathbb{Z}_q$
  store $sk_{\mathcal{X}} \leftarrow (esk_{\mathcal{X}}, \{x_A, ssk_{\mathcal{X},A}\}_{\forall \mathcal{S}_A \in \mathbf{S}})$
  register $pk_{\mathcal{X}} \leftarrow (epk_{\mathcal{X}}, \{spk_{\mathcal{X},A}\}_{\forall \mathcal{S}_A \in \mathbf{S}})$ with $\mathcal{F}_{\mathsf{CA}}$

Server Setup (by each server $\mathcal{S}_A \in \mathbf{S}$):
  $(epk_A, esk_A) \xleftarrow{\$} \mathsf{EncKGen}_{\mathbb{G}}(1^\tau)$
  $k_A \xleftarrow{\$} \mathsf{PRPKGen}_{\mathbb{G}}(1^\tau)$
  store $sk_A \leftarrow (esk_A, k_A)$
  register $pk_A \leftarrow epk_A$ with $\mathcal{F}_{\mathsf{CA}}$

User Setup (by each user $\mathcal{U}_i$):
  $(upk_i \in \mathbb{G}^n, usk_i) \xleftarrow{\$} \mathsf{REncKGen}_{\mathbb{G}}(1^\tau)$
  choose a random $uid_i \xleftarrow{\$} \mathbb{Z}_q$
  store $(usk_i, upk_i, uid_i)$
```

Figure 7: Setup of Converter, Servers, and Users.

**4.1.2. Pseudonym Generation.** When a user $\mathcal{U}_i$ wants to generate a pseudonym $nym_{i,A}$ towards a server $\mathcal{S}_A$, $\mathcal{U}_i$ and $\mathcal{X}$ first jointly prepare the pseudonym stub. In a second step, $\mathcal{X}$ and $\mathcal{S}_A$ complete the pseudonym generation. Both protocols are depicted in Figure 8, we specify the proof $\pi_{\mathcal{U}}$ and used enc/decoding functions below.

The user has to prove in $\pi_{\mathcal{U}}$ that he correctly encrypted the output from $\mathcal{F}_{\mathsf{cOPRF}}$ and that his contributions to the audit tag $T$ are correct:

$$\begin{aligned}
&\pi_{\mathcal{U}} \xleftarrow{\$} \mathrm{NIZK}\{(\underline{z_i}, \underline{r_1}, o_z) : \mathsf{ComVf}_{\mathbb{G}}(z_i, com_z, o_z) = 1 \quad \wedge \\
&C_{nym} = \mathsf{Enc}_{\mathbb{G}}(epk_A, z_i) \quad \wedge \quad C_r = \mathsf{Enc}_{\mathbb{G}}(epk_A, r_1) \quad \wedge \\
&C_r^* = \mathsf{REnc}_{\mathbb{G}}(upk_i', r_1)\}(sid, nqid, C_{nym}, C_r, C_r^*, upk_i', com_z)
\end{aligned}$$

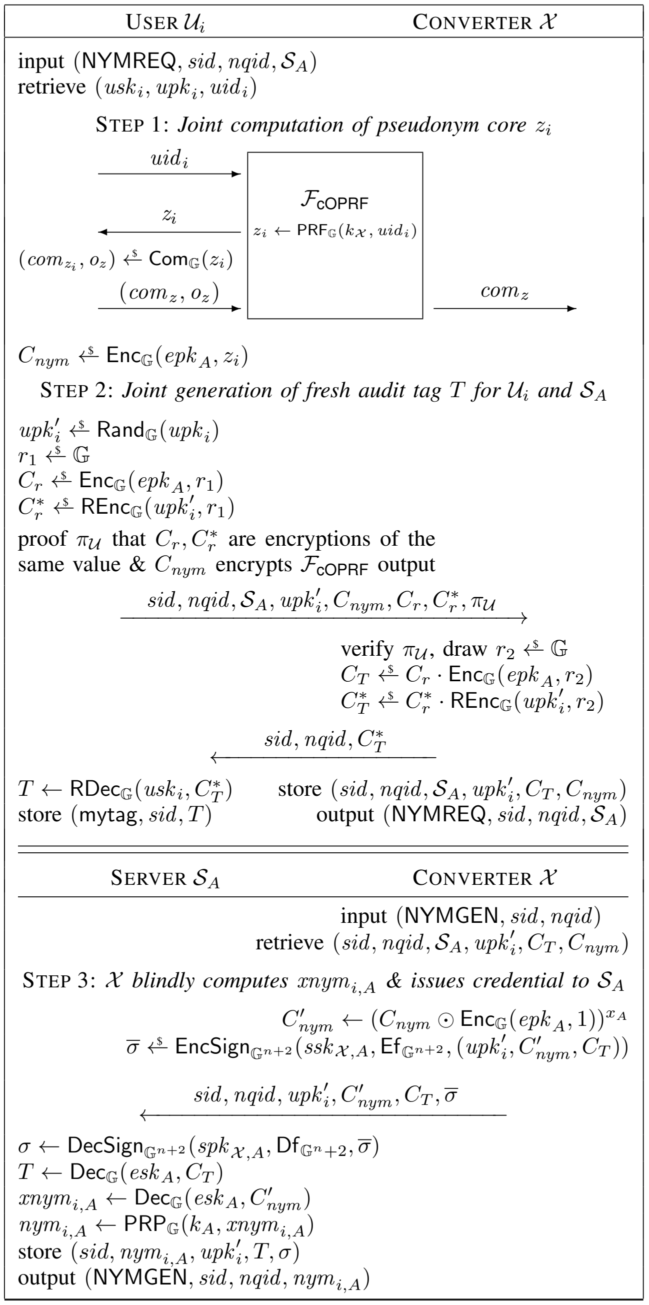
Binding the proof to all ciphertexts, session identifiers and $upk_i'$, guarantees that an adversary cannot "impersonate" an honest user by re-using his encrypted core-pseudonym $z_i$ in another session or replace his key $upk_i'$.

When the converter completes the pseudonym towards $\mathcal{S}_A$, it signs a mixed block of plaintext and encrypted messages using our signature scheme for encoded messages w.r.t. the following functions:

$$\begin{aligned}
&\mathsf{Ef}_{\mathbb{G}^{n+2}}(m_1, m_2, m_3) = \\
&\qquad\qquad (m_1, \mathsf{Enc}_{\mathbb{G}}(epk_A, m_2), \mathsf{Enc}_{\mathbb{G}}(epk_A, m_3)) \\
&\mathsf{Df}_{\mathbb{G}^{n+2}}(c_1, c_2, c_3) = (c_1, \mathsf{Dec}_{\mathbb{G}}(esk_A, c_2), \mathsf{Dec}_{\mathbb{G}}(esk_A, c_3)).
\end{aligned}$$

Both functions work in group $\mathbb{G}^{n+2} = \mathbb{G}^n \times \mathbb{G} \times \mathbb{G}$, where $\mathbb{G}^n$ is the public key space of $\mathsf{REnc}_{\mathbb{G}}$ and $\mathbb{G}$ the domain of the encryption scheme $\mathsf{Enc}_{\mathbb{G}}$. As in the CL-15 protocol, the converter has a dedicated signing key for each server which is crucial to ensure that only the server $\mathcal{S}_A$, for which the pseudonym was intended for, can subsequently use it in a conversion request.

**4.1.3. Pseudonym Conversion.** When a server $\mathcal{S}_A$ wishes to convert a pseudonym $nym_{i,A}$ towards a server $\mathcal{S}_B$, it sends a conversion request to $\mathcal{X}$. If the request is legitimate, the converter and $\mathcal{S}_B$ then jointly complete the conversion.



User $\mathcal{U}_i$ — Converter $\mathcal{X}$

input $(\mathsf{NYMREQ}, sid, nqid, \mathcal{S}_A)$
retrieve $(usk_i, upk_i, uid_i)$

Step 1: *Joint computation of pseudonym core $z_i$*

$\xrightarrow{uid_i}$ $\boxed{\mathcal{F}_{\mathsf{cOPRF}} \quad z_i \leftarrow \mathsf{PRF}_{\mathbb{G}}(k_{\mathcal{X}}, uid_i)}$
$\xleftarrow{z_i}$
$(com_{z_i}, o_z) \xleftarrow{\$} \mathsf{Com}_{\mathbb{G}}(z_i)$
$\xrightarrow{(com_z, o_z)}$ $\xrightarrow{com_z}$

$C_{nym} \xleftarrow{\$} \mathsf{Enc}_{\mathbb{G}}(epk_A, z_i)$

Step 2: *Joint generation of fresh audit tag $T$ for $\mathcal{U}_i$ and $\mathcal{S}_A$*

$upk'_i \xleftarrow{\$} \mathsf{Rand}_{\mathbb{G}}(upk_i)$
$r_1 \xleftarrow{\$} \mathbb{G}$
$C_r \xleftarrow{\$} \mathsf{Enc}_{\mathbb{G}}(epk_A, r_1)$
$C^*_r \xleftarrow{\$} \mathsf{REnc}_{\mathbb{G}}(upk'_i, r_1)$
proof $\pi_{\mathcal{U}}$ that $C_r, C^*_r$ are encryptions of the same value & $C_{nym}$ encrypts $\mathcal{F}_{\mathsf{cOPRF}}$ output

$\xrightarrow{sid, nqid, \mathcal{S}_A, upk'_i, C_{nym}, C_r, C^*_r, \pi_{\mathcal{U}}}$

verify $\pi_{\mathcal{U}}$, draw $r_2 \xleftarrow{\$} \mathbb{G}$
$C_T \xleftarrow{\$} C_r \cdot \mathsf{Enc}_{\mathbb{G}}(epk_A, r_2)$
$C^*_T \xleftarrow{\$} C^*_r \cdot \mathsf{REnc}_{\mathbb{G}}(upk'_i, r_2)$

$\xleftarrow{sid, nqid, C^*_T}$

$T \leftarrow \mathsf{RDec}_{\mathbb{G}}(usk_i, C^*_T)$ — store $(sid, nqid, \mathcal{S}_A, upk'_i, C_T, C_{nym})$
store $(\mathsf{mytag}, sid, T)$ — output $(\mathsf{NYMREQ}, sid, nqid, \mathcal{S}_A)$

Server $\mathcal{S}_A$ — Converter $\mathcal{X}$

input $(\mathsf{NYMGEN}, sid, nqid)$
retrieve $(sid, nqid, \mathcal{S}_A, upk'_i, C_T, C_{nym})$

Step 3: *$\mathcal{X}$ blindly computes $xnym_{i,A}$ & issues credential to $\mathcal{S}_A$*

$C'_{nym} \leftarrow (C_{nym} \odot \mathsf{Enc}_{\mathbb{G}}(epk_A, 1))^{x_A}$
$\overline{\sigma} \xleftarrow{\$} \mathsf{EncSign}_{\mathbb{G}^{n+2}}(ssk_{\mathcal{X},A}, \mathsf{Ef}_{\mathbb{G}^{n+2}}, (upk'_i, C'_{nym}, C_T))$

$\xleftarrow{sid, nqid, upk'_i, C'_{nym}, C_T, \overline{\sigma}}$

$\sigma \leftarrow \mathsf{DecSign}_{\mathbb{G}^{n+2}}(spk_{\mathcal{X},A}, \mathsf{Df}_{\mathbb{G}^n+2}, \overline{\sigma})$
$T \leftarrow \mathsf{Dec}_{\mathbb{G}}(esk_A, C_T)$
$xnym_{i,A} \leftarrow \mathsf{Dec}_{\mathbb{G}}(esk_A, C'_{nym})$
$nym_{i,A} \leftarrow \mathsf{PRP}_{\mathbb{G}}(k_A, xnym_{i,A})$
store $(sid, nym_{i,A}, upk'_i, T, \sigma)$
output $(\mathsf{NYMGEN}, sid, nqid, nym_{i,A})$

Figure 8: Pseudonym Request & Generation Protocol.

To distinguish multiple conversion requests and give the servers a handle to identify the session, each request is associated to a unique query identifier $cqid = (\mathcal{S}_A, \mathcal{S}_B, cqid')$. The conversion request and response protocols are depicted in Figure 9 and we specify the two proofs $\pi_A$ and $\pi_B$ below.

When making a conversion request, $\mathcal{S}_A$ has to prove in $\pi_A$ that all its contributions are correct and consistent. In particular, it has to show that it encrypted the same $xnym_{i,A}$ in $C_{nym,A}$ and $C_{nym,B}$, where the former ciphertext is used to let $\mathcal{X}$ blindly re-issue a new one-time credential to $\mathcal{S}_A$, and $C_{nym,B}$ will be blindly transformed into the pseudonym for $\mathcal{S}_B$. $\mathcal{S}_A$ further has to show that his contribution to the new random audit tag $T_A$ are correct and the user-specific encryption is done for a correctly re-randomized public key $upk'$. That all computations are made for legitimate values is guaranteed by proving knowledge of a signature $\sigma$ on the encrypted messages. Overall, $\pi_A$ looks as follows:

$$\begin{aligned}\pi_A \xleftarrow{\$} \text{NIZK}\{(&\underline{upk}, \underline{xnym_{i,A}}, \underline{\sigma}, \underline{r}) : \\ &\mathsf{Vf}_{\mathbb{G}}(spk_{\mathcal{X},\mathcal{A}}, \sigma, (upk, xnym_{i,A}, T)) = 1 \quad \wedge \\ &C_{nym,A} = \mathsf{Enc}_{\mathbb{G}}(epk_A, xnym_{i,A}) \quad \wedge \\ &C_{nym,B} = \mathsf{Enc}_{\mathbb{G}}(epk_B, xnym_{i,A}) \quad \wedge \\ C_r = \mathsf{Enc}_{\mathbb{G}}(&epk_A, r) \quad \wedge \quad C^*_r = \mathsf{REnc}_{\mathbb{G}}(upk', r) \quad \wedge \\ &upk' = \mathsf{Rand}_{\mathbb{G}}(upk) \\ &\}(sid, cqid, C_{nym,A}, C_{nym,B}, C_r, C^*_r, upk', T).\end{aligned}$$

When the converter approves the request, it blindly computes $xnym_{i,B} \leftarrow xnym_{i,A}^{x_B/x_A}$ and signs the encrypted pseudonym together with the randomized user key and encrypted tag $T_B$. The tag has been jointly computed with $\mathcal{S}_B$ who has to prove correctness of his contribution in a proof $\pi_B$:

$$\begin{aligned}\pi_B \xleftarrow{\$} \text{NIZK}\{(\underline{r}) : C_r = \mathsf{Enc}_{\mathbb{G}}(epk_A, r) \quad \wedge \\ C^*_r = \mathsf{REnc}_{\mathbb{G}}(upk', r)\}(sid, cqid, C_r, C^*_r, ).\end{aligned}$$

**4.1.4. User Audit.** The audit procedure allows a user $\mathcal{U}_i$ to learn all conversion requests related to his pseudonyms. Therefore $\mathcal{U}_i$ first uses the audit tags $T_j$ he received at each pseudonym generation to iteratively retrieve the chain of new audit tags that were generated whenever his pseudonyms got converted. These new audit tags are published in encrypted form at the bulletin board and are associated with the "old" tag that was used for the conversion. When $\mathcal{U}_i$ has retrieved all his audit tags, he decrypts for each tag the corresponding audit log entry containing the query identifier $cqid$. Recall that these identifiers have the form $cqid = (cqid', \mathcal{S}_A, \mathcal{S}_B)$, i.e., specify both servers involved in the conversion request. The detailed audit procedure is given in Figure 10.

We opted for full download of the bulletin board for the sake of simplicity and best privacy guarantees. Of course, a user would not have to download all entries at every time, but could only retrieve the newest ones (sacrificing some privacy though). In fact, as the audit procedure can be done entirely one-the-fly, the users could retrieve the bulletin board in form of an RSS feeds. When receiving an update, each user would check for log entries that are labeled with one of his tags, process these entries and delete the unrelated ones. Another alternative that requires less communication is the use of private-information retrieval, but which in turn increases the computational costs.

## 5. Security

We now show that our auditable pseudonym system presented in the previous section securely realizes the ideal functionality $\mathcal{F}^{\mathsf{leak}}_{\text{nym-log}}$ against actively corrupted users and servers, and passively corrupted converter.

```
SERVER $\mathcal{S}_A$                                   CONVERTER $\mathcal{X}$

input $(\mathsf{CONVERT}, sid, cqid, nym_{i,A})$
retrieve $(sid, nym_{i,A}, upk, T, \sigma)$

        STEP 1: *Conversion request w.r.t. tag $T$*

$upk' \xleftarrow{\$} \mathsf{Rand}_{\mathbb{G}}(upk)$
$xnym_{i,A} \leftarrow \mathsf{PRP}_{\mathbb{G}}^{-1}(k_A, nym_{i,A})$
$C_{nym,A} \xleftarrow{\$} \mathsf{Enc}_{\mathbb{G}}(epk_A, xnym_{i,A})$
$C_{nym,B} \xleftarrow{\$} \mathsf{Enc}_{\mathbb{G}}(epk_B, xnym_{i,A})$
$r_1 \xleftarrow{\$} \mathbb{G}$
$C_r \xleftarrow{\$} \mathsf{Enc}_{\mathbb{G}}(epk_A, r_1)$
$C_r^* \xleftarrow{\$} \mathsf{REnc}_{\mathbb{G}}(upk', r_1)$
proof $\pi_A$ that all $C$'s and $upk'$ are
computed correctly, and are based on
values for which $\mathcal{S}_A$ has a valid $\sigma$

        $\xrightarrow{sid,\ cqid,\ upk',\ C_{nym,A},\ C_{nym,B},\ C_r,\ C_r^*,\ T,\ \pi_A}$

        STEP 2: *$\mathcal{X}$ completes generation of fresh audit tag $T_A$ for $\mathcal{S}_A$
        & publishes encryption of new tag under $upk'$*

                                          verify $\pi_A$ and that $T \notin \mathbf{L}_T$
                                          update $\mathbf{L}_T \leftarrow \mathbf{L}_T \cup T$
                                          $r_2 \xleftarrow{\$} \mathbb{G}$
                                          $C_{T,A} \xleftarrow{\$} C_r \cdot \mathsf{Enc}_{\mathbb{G}}(epk_A, r_2)$
                                          $C_{T,A}^* \xleftarrow{\$} C_r^* \cdot \mathsf{REnc}_{\mathbb{G}}(upk', r_2)$
                                          $C_{\log}^* \xleftarrow{\$} \mathsf{REnc}_{\mathbb{G}}(upk', cqid)$
                                          publish in audit log $\mathcal{F}_{\mathsf{BB}}$:
                                          $(\mathsf{convert}, sid, T, C_{\log}^*, \mathtt{request})$
                                          and $(\mathsf{audittag}, sid, T, C_{T,A}^*)$

        STEP 3: *$\mathcal{X}$ blindly issues new one-time credential to $\mathcal{S}_A$*

        $\overline{\sigma}_A \leftarrow \mathsf{EncSign}_{\mathbb{G}^{n+2}}(ssk_{\mathcal{X},A}, \mathsf{Ef}_{\mathbb{G}^{n+2}}, (upk', C_{nym,A}, C_{T,A}))$
        $\xleftarrow{sid,\ cqid,\ C_{T,A},\ \overline{\sigma}_A}$

$\sigma_A \leftarrow \mathsf{DecSign}_{\mathbb{G}^{n+2}}(spk_{\mathcal{X},A}, \mathsf{Df}_{\mathbb{G}^{n+2}}, \overline{\sigma}_A)$
$T_A \leftarrow \mathsf{Dec}_{\mathbb{G}}(esk_A, C_{T,A})$
store $(sid, nym_{i,A}, upk', T_A, \sigma_A)$     store $(sid, cqid, C_{nym,B}, upk', T)$
delete $(sid, nym_{i,A}, upk, T, \sigma)$         output $(\mathsf{CONVERT}, sid, cqid)$
```

```
CONVERTER $\mathcal{X}$                                  SERVER $\mathcal{S}_B$

input $(\mathsf{PROCEED}, sid, cqid)$
retrieve $(sid, cqid, C_{nym,B}, upk', T)$
publish in audit log $\mathcal{F}_{\mathsf{BB}}$:
$(\mathsf{convert}, sid, T, \mathtt{done})$

        $\xrightarrow{sid,\ cqid,\ upk'}$

        STEP 4: *Joint generation of fresh audit tag $T_B$ for $\mathcal{S}_B$
        & $\mathcal{X}$ publishes encryption of new tag under $upk'$*

                                          $r_1 \xleftarrow{\$} \mathbb{G}$
                                          $C_r \xleftarrow{\$} \mathsf{Enc}_{\mathbb{G}}(epk_B, r_1)$
                                          $C_r^* \xleftarrow{\$} \mathsf{REnc}_{\mathbb{G}}(upk', r_1)$
                                          proof $\pi_B$ that $C_r, C_r^*$ are en-
                                          cryptions of the same value

        $\xleftarrow{sid,\ cqid,\ C_r,\ C_r^*,\ \pi_B}$

verify $\pi_B$
$r_2 \xleftarrow{\$} \mathbb{G}$
$C_{T,B} \xleftarrow{\$} C_r \cdot \mathsf{Enc}_{\mathbb{G}}(epk_B, r_2)$
$C_{T,B}^* \xleftarrow{\$} C_r^* \cdot \mathsf{REnc}_{\mathbb{G}}(upk', r_2)$
publish in audit log $\mathcal{F}_{\mathsf{BB}}$:
$(\mathsf{audittag}, sid, T, C_{T,B}^*)$

        STEP 5: *$\mathcal{X}$ converts pseudonym
        & blindly issues one-time credential to $\mathcal{S}_B$*

$\Delta \leftarrow x_B / x_A \pmod q$
$C'_{nym,B} \leftarrow (C_{nym,B} \odot \mathsf{Enc}_{\mathbb{G}}(epk_B, 1))^{\Delta}$
$\overline{\sigma}_B \leftarrow \mathsf{EncSign}_{\mathbb{G}^{n+2}}(ssk_{\mathcal{X},B}, \mathsf{Ef}_{\mathbb{G}^{n+2}}, (upk', C'_{nym,B}, C_{T,B}))$

        $\xrightarrow{sid,\ cqid,\ C'_{nym,B},\ C_{T,B},\ \overline{\sigma}_B}$

                    $\sigma_B \leftarrow \mathsf{DecSign}_{\mathbb{G}^{n+2}}(spk_{\mathcal{X},B}, \mathsf{Df}_{\mathbb{G}^{n+2}}, \overline{\sigma}_B)$
                                          $T_B \leftarrow \mathsf{Dec}_{\mathbb{G}}(esk_B, C_{T,B})$
                                          $xnym_{i,B} \leftarrow \mathsf{Dec}_{\mathbb{G}}(esk_B, C'_{nym,B})$
                                          $nym_{i,B} \leftarrow \mathsf{PRP}_{\mathbb{G}}(k_B, xnym_{i,B})$
                                          store $(sid, nym_{i,B}, upk', T_B, \sigma_B)$
                                          output $(\mathsf{CONVERTED}, sid, cqid, nym_{i,B})$
```

Figure 9: Conversion Protocol (left: Request, right: Response).

***Theorem 5.1.*** Our auditable pseudonym system described in Section 4 securely implements the ideal functionality $\mathcal{F}^{\mathsf{leak}}_{\text{nym-log}}$ defined in Section 2 (with leak being a deterministic one-way function), against mixed adversaries described below and in the $(\mathcal{F}_{\mathsf{CA}}, \mathcal{F}_{\mathsf{CRS}}, \mathcal{F}_{\mathsf{AUTH}}, \mathcal{F}_{\mathsf{1\text{-}AUTH}}, \mathcal{F}_{\mathsf{BB}}, \mathcal{F}^{\mathsf{PRF,Com}}_{\mathsf{cOPRF}})$ hybrid-model, provided that

- $(\mathsf{EncKGen}_{\mathbb{G}}, \mathsf{Enc}_{\mathbb{G}}, \mathsf{Dec}_{\mathbb{G}})$ is a secure homomorphic encryption scheme,
- $(\mathsf{REncKGen}_{\mathbb{G}}, \mathsf{Rand}_{\mathbb{G}}, \mathsf{REnc}_{\mathbb{G}}, \mathsf{RDec}_{\mathbb{G}})$ is a secure homomorphic key-randomizable encryption scheme,
- $(\mathsf{SigKGen}_{\mathbb{G}^{n+2}}, \mathsf{EncSign}_{\mathbb{G}^{n+2}}, \mathsf{DecSign}_{\mathbb{G}^{n+2}}, \mathsf{Vf}_{\mathbb{G}^{n+2}})$ is an unforgeable signature scheme for encoded messages,
- $(\mathsf{PRFKGen}_{\mathbb{G}}, \mathsf{PRF}_{\mathbb{G}})$ is a secure and one-way PRF,
- $(\mathsf{PRPKGen}_{\mathbb{G}}, \mathsf{PRP}_{\mathbb{G}})$ is a secure PRP,
- $(\mathsf{Com}_{\mathbb{G}}, \mathsf{ComVf}_{\mathbb{G}})$ is a secure commitment scheme,
- the proof system used for NIZK is zero-knowledge, simulation-sound and online-extractable (for the underlined values), and
- the DDH-assumption holds in group $\mathbb{G}$.

We discuss the considered mixed adversary type below, and provide the proof of Theorem 5.1 in the full version of this paper.

*Mixed Adversaries.* To prove our protocol secure we consider a *mixed* adversary [13] that can actively corrupt servers and users but only gain passive control over the converter. We consider this a realistic model: for servers and users which can have diverse and hard to control backgrounds we do not make any assumptions on their trustworthiness and tolerate entirely malicious behaviour. For the converter, who is the central and crucial entity in our system, we assume that correct behaviour can be controlled and enforced more easily which allows to consider only honest-but-curious adversaries. That is, even when corrupted, the converter will perform the protocol correctly. This avoids the complexity needed to cryptographically enforce $\mathcal{X}$'s compliance, while it still covers the main threats: an overly curious converter trying to trace or identify users by exploiting all the information and keys it sees, or $\mathcal{X}$ getting compromised by an attacker stealing all internal data.

User $\mathcal{U}_i$ upon input $(\mathsf{AUDIT}, sid)$ retrieves and decrypts his logs from the bulletin board $\mathcal{F}_{\mathsf{BB}}$:

STEP 1: *Retrieve all data from bulletin board.*
Retrieve all audit tag entries $\{(\mathsf{audittag}, sid, T_k, C_T^*)\}$ and conversion records $\{(\mathsf{convert}, sid, T_k, C_{\text{log}}^*, \texttt{request})\}$ and $\{(\mathsf{convert}, sid, T_k, \mathsf{done})\}$ for $sid$ from the bulletin board $\mathcal{F}_{\mathsf{BB}}$.

STEP 2: *Recursively get chain of audit tags.*
Retrieve all locally stored audit tags $(\mathsf{mytag}, sid, T_j)$, and for each $T_j$ do the following:

a) Look up all related tag entries $(\mathsf{audittag}, sid, T_j, C_T^*)$ for $T_j$.
b) Decrypt the follow-up audit tag as $T_j' \leftarrow \mathsf{RDec}_{\mathbb{G}}(usk_i, C_T^*)$.
c) Store $(\mathsf{mytag}, sid, T_j')$ and repeat procedure for $T_j'$.

STEP 3: *Decrypt conversion logs for audit tags.*
Retrieve all stored tags $(\mathsf{mytag}, sid, T_j)$, set $\mathbf{L}_{\text{log}} \leftarrow \emptyset$, and for each $T_j$ do the following:

a) Look up entry $(\mathsf{convert}, sid, T_j, C_{\text{log}}^*, \texttt{request})$ for $T_j$.
b) Decrypt audit info as $cqid_j \leftarrow \mathsf{RDec}_{\mathbb{G}}(usk_i, C_{\text{log}}^*)$.
c) If a record $(\mathsf{convert}, sid, T_j, \mathsf{done})$ for $T_j$ exists as well, set $status_j \leftarrow \texttt{done}$ and $status_j \leftarrow \texttt{request}$ otherwise.
d) Update $\mathbf{L}_{\text{log}} \leftarrow \mathbf{L}_{\text{log}} \cup \{(cqid_j, status_j)\}$.

End with output $(\mathsf{AUDIT}, sid, \mathbf{L}_{\text{log}})$.

Figure 10: User Audit.

This mixed setting of actively and passively corrupted parties then requires some (often only implicitly handled) trust assumptions in order to maintain the difference between active and passive corruption. Recall that actively corrupted parties are fully controlled by the adversary, whereas a passively corrupted entity still behaves honestly but the adversary can see all its inputs, outputs and internal state. However, the adversary can actually not be allowed to learn the full state including *all* secret keys of a passively corrupted party $\mathcal{P}$. Otherwise, $\mathcal{A}$ could simply impersonate $\mathcal{P}$ by running an identical copy which the adversary then actively controls. Thus, to enforce $\mathcal{A}$'s passive behaviour for $\mathcal{P}$, one assumes authenticated channels that remain intact even upon passive corruption which prevents such "impersonation attacks" [28], [16].

Consequently, an adversary passively corrupting the converter in our protocol will not receive $\mathcal{X}$'s secret keys used to realize $\mathcal{F}_{\mathsf{AUTH}}$, $\mathcal{F}_{\mathsf{1\text{-}AUTH}}$, and $\mathcal{F}_{\mathsf{BB}}$. Moreover, as the signature scheme on encoded messages is used by the converter to issue credentials which can be used in an anonymous manner, we assume that the corresponding secret key cannot be exposed upon corruption either. In fact, giving these keys to the adversary would no longer reflect honest-but-curious behaviour of $\mathcal{X}$: the adversary could sign arbitrary combinations of pseudonyms, public keys and tags on behalf of $\mathcal{X}$, e.g., irregularly mixing pseudonyms of honest users with adversarially controlled public keys. When the adversary also actively corrupted some servers, he then can circulate these malicious signatures, thereby indirectly impersonating $\mathcal{X}$. Thus, we grant the signing keys the same necessary protection as the authentication keys, which could e.g., be realized via HSM's.

An adversary passively corrupting $\mathcal{X}$ still gets to see all internal protocol messages as well as the secret PRF key $k_{\mathcal{X}}$ and all secret exponents $x_A, x_B, \ldots$ used to generate and convert pseudonyms. For actively corrupted users and servers, of course all keys are known to the adversary (as he generates them).

## 6. Concrete Instantiation

Our protocol can be instantiated with the ElGamal encryption scheme with CRS trapdoor for $(\mathsf{EncKGen}_{\mathbb{G}}, \mathsf{Enc}_{\mathbb{G}}, \mathsf{Dec}_{\mathbb{G}})$, the R-ElGamal scheme for $(\mathsf{REncKGen}_{\mathbb{G}}, \mathsf{REnc}_{\mathbb{G}}, \mathsf{RDec}_{\mathbb{G}}, \mathsf{Rand}_{\mathbb{G}})$, the Gr+ for the signature scheme for encoded messages $(\mathsf{SigKGen}_{\mathbb{G}}, \mathsf{Sign}_{\mathbb{G}}, \mathsf{EncSign}_{\mathbb{G}}, \mathsf{DecSign}_{\mathbb{G}}, \mathsf{Vf}_{\mathbb{G}})$ and the $\Pi_{\mathsf{cOPRF}}$ protocol for $\mathcal{F}_{\mathsf{cOPRF}}$. For the $\mathsf{PRP}_{\mathbb{G}}$ we use lazy sampling as in [7], symmetric encryption is an alternative as well, with proper mapping from $\mathbb{G}$ into the domain and from the range of the symmetric cipher.

### 6.1. NIZK Instantiations

We now provide the concrete instantiations of the non-interactive proofs when our scheme is instantiated as described above. A number of secrets of which knowledge is proven need to be online extractable. To achieve this, we employ for the encryption scheme $\mathsf{Enc}_{\mathbb{G}}(\cdot,\cdot)$ the ElGamal scheme extended with a CRS trapdoor, as discussed in Section 3.4, thus, e.g., $C = \mathsf{Enc}_{\mathbb{G}}(epk, m, \rho) = (C_0, C_1, C_2) = (\hat{y}^\rho, epk^\rho, g^\rho m)$ and use for $\mathsf{REnc}_{\mathbb{G}}(upk, m, \rho) = C = (C_1, C_2) = (upk_1^\rho, upk_2^\rho m)$.

*Pseudonym Generation.* We first consider the following proof $\pi_{\mathcal{U}}$, with which the user proves that he correctly encrypted the output obtained from $\mathcal{F}_{\mathsf{cOPRF}}$ (to which the verifier had obtained the commitment $com_z$) and that $C_r$ and $C_r^*$, which will be used to derive the audit tag, are encryptions of the same value $r$. Let $com_z = (com_{z,1}, com_{z,2}) = (h^{o_z}, g^{o_z} z_i)$ be the concrete commitment to $z_i$. Then $\pi_{\mathcal{U}}$ is as follows.

$$\begin{aligned}\pi_{\mathcal{U}} \xleftarrow{\$} \mathrm{SPK}\{\rho_1, \rho_2, \rho_3, o_z) :& \\ C_{nym,0} = \hat{y}^{\rho_3} \ \wedge\ C_{nym,1} = epk_A^{\rho_3} \ \wedge& \\ com_{z,1} = h^{o_z} \ \wedge\ com_{z,2}/C_{nym,2} = g^{o_z} g^{-\rho_3}& \\ C_{r,0} = \hat{y}^{\rho_1} \ \wedge\ C_{r,1} = epk_A^{\rho_1} \ \wedge\ C_{r,1}^* = {upk'_1}^{\rho_2} \ \wedge& \\ C_{r,2}/C_{r,2}^* = g^{\rho_1} {upk'_2}^{-\rho_2} \ \wedge& \\ \}(sid, nqid, C_{nym}, C_r, C_r^*, com_z) \ .&\end{aligned}$$

*Pseudonym Conversion.* We next consider the proof $\pi_A$ that all $C_{nym,A}$, $C_{nym,B}$, $C_r$, $C_r^*$ and $upk'$ were computed correctly and based on values that were signed. To provide the concrete instantiation, let $upk = (upk_1, upk_2) \in \mathbb{G}^2$, $xnym_{i,A} \in \mathbb{G}$, and $T \in \mathbb{G}$. Next, let $\sigma = (\tilde{r}, \tilde{s}, \tilde{t}_1, \tilde{t}_2, \tilde{t}_3, \tilde{t}_4)$ be a freshly randomized signature on $(upk_1, upk_2, xnym_{i,A}, T) \in \mathbb{G}^4$. Finally, let $upk' = (upk'_1, upk'_2) = \mathsf{Rand}_{\mathbb{G}}(upk, \rho_5) = (upk_1^{1/\rho_5}, upk_2^{1/\rho_5})$ for a random $\rho_5 \xleftarrow{\$} \mathbb{Z}_q$. As part of the proof, the prover provides the two randomized signature values $\tilde{r}$ and $\tilde{s}$ and encryptions of $upk_1$, $upk_2$, and all $\tilde{t}_i$'s under the public key $\hat{y}$ contained in the CRS. That is, let $\tilde{T}_i = (\tilde{T}_{i,1}, \tilde{T}_{i,2}) = (\hat{y}^{\nu_i}, g^{\nu_i}\tilde{t}_i)$, for random $\nu_i \xleftarrow{\$} \mathbb{Z}_q$, be encryptions of the $\tilde{t}_i$ values and let $(U_{i,1}, U_{i,2}) = (\hat{y}^{\xi_i}, g^{\xi_i} upk_i)$ be encryptions of $upk_i$ for random $\xi_i \xleftarrow{\$} \mathbb{Z}_q$. Then the prover computes

$$
\begin{gathered}
\pi_A \xleftarrow{\$} \mathrm{SPK}\{(\tilde{t}_1, \ldots, \tilde{t}_4, r, \xi_1, \xi_2, \nu_1, \ldots, \nu_4, \rho_1, \ldots, \rho_5) : \\
e(x_1, y)e(U_{1,2}, \tilde{g})/e(\tilde{T}_{1,2}, \tilde{r}) = e(g, \tilde{r})^{-\nu_1} e(g, \tilde{g})^{\xi_1} \ \wedge \\
e(x_2, y)e(U_{2,2}, \tilde{g})/e(\tilde{T}_{2,2}, \tilde{r}) = e(g, \tilde{r})^{-\nu_2} e(g, \tilde{g})^{\xi_2} \ \wedge \\
e(x_3, y)e(C_{nym,A,2}, \tilde{g})/e(\tilde{T}_{3,2}, \tilde{r}) = e(g, \tilde{r})^{-\nu_3} e(g, \tilde{g})^{\rho_1} \ \wedge \\
e(x_4, y)e(T, \tilde{g})/e(\tilde{T}_{4,2}, \tilde{r}) = e(g, \tilde{r})^{-\nu_4} \ \wedge \\
\tilde{T}_{1,1} = \hat{y}^{\nu_1} \ \wedge \ \tilde{T}_{2,1} = \hat{y}^{\nu_2} \ \wedge \ \tilde{T}_{3,1} = \hat{y}^{\nu_3} \ \wedge \ \tilde{T}_{4,1} = \hat{y}^{\nu_4} \ \wedge \\
C_{nym,A,0} = \hat{y}^{\rho_1} \quad \wedge \quad C_{nym,B,0} = \hat{y}^{\rho_2} \quad \wedge \\
C_{nym,A,1} = epk_A^{\rho_1} \ \wedge \ C_{nym,B,1} = epk_B^{\rho_2} \ \wedge \\
C_{nym,A,2}/C_{nym,B,2} = g^{\rho_1} g^{-\rho_2} \ \wedge \\
C_{r,0} = \hat{y}^{\rho_3} \ \wedge \ C_{r,1} = epk_A^{\rho_3} \ \wedge \ C^*_{r,1} = {upk'_1}^{\rho_4} \ \wedge \\
C_{r,2}/C^*_{r,2} = g^{\rho_3} {upk'_2}^{-\rho_4} \quad \wedge \\
U_{1,1} = \hat{y}^{\xi_1} \ \wedge \ U_{2,1} = \hat{y}^{\xi_2} \ \wedge \ U_{1,2} = g^{\xi_1} {upk'_1}^{\rho_5} \ \wedge \\
U_{2,2} = g^{\xi_2} {upk'_2}^{\rho_5}\}(sid, cqid, e, g, \tilde{g}, x, y, \tilde{r}, \tilde{s}, T, \bar{y}, \\
\tilde{T}_1, \ldots, \tilde{T}_4, U_1, U_2, C_{nym,A}, C_{nym,B}, C_r, C^*_r).
\end{gathered}
$$

To verify the proof, the verifier checks whether $e(g, y)e(x_1, \tilde{g}) = e(\tilde{s}, \tilde{r})$ holds and whether $\pi_A$ is a correct proof of the statement above.

Finally, deriving the proof $\pi_B$ of showing that $C_r$ and $C^*_r$ encrypt the same value is easy:

$$
\begin{aligned}
\pi_B \xleftarrow{\$} \ \mathrm{SPK}\{(\rho_1, \rho_2) :\ & C_{r,0} = \hat{y}^{\rho_1} \ \wedge \ C_{r,1} = epk_A^{\rho_1} \ \wedge \\
& C^*_{r,1} = {upk'_1}^{\rho_2} \ \wedge \ C_{r,2}/C^*_{r,2} = g^{\rho_1} {upk'_2}^{-\rho_2} \\
& \}(sid, cqid, C_r, C^*_r).
\end{aligned}
$$

## 6.2. Security & Efficiency

If our scheme is instantiated with the NIZK's and building blocks stated above, then by the security of the building blocks, our scheme is secure in the $(\mathcal{F}_{\mathsf{CA}}, \mathcal{F}_{\mathsf{CRS}}, \mathcal{F}_{\mathsf{AUTH}}, \mathcal{F}_{\mathsf{1\text{-}AUTH}}, \mathcal{F}_{\mathsf{BB}})$-hybrid model under the Symmetric eXternal Decision Diffie-Hellman (SXDH) assumption [2], the $q$-Decisional Diffie-Hellman Inversion assumption [5], the Decision Composite Residuosity assumption [19] and the unforgeability of Groth's signature scheme [15].

With the instantiation described above we get the following efficiency figures, specifying the amount of exponentiations and pairings in the respective groups. We assume obvious optimizations such as making use of multi-exponentiations and multi-pairings, or computing exponentiations in $\mathbb{G}$ instead of $\mathbb{G}_t$ whenever applicable. The computations in group $\mathbb{Z}^*_{n^2}$ with $n$ being a RSA-modulus stem from the Camenisch-Shoup encryption used in the realization of $\mathcal{F}_{\mathsf{cOPRF}}$.

| | | | |
|---|---|---|---|
| Generation : | $\mathcal{U}_i$ | $\mathcal{X}$ | $\mathcal{S}_A$ |
| | $22\mathbb{G} + 6\mathbb{Z}^*_{n^2}$ | $22\mathbb{G} + \tilde{\mathbb{G}} + 6\mathbb{Z}^*_{n^2}$ | $4\mathbb{G}$ |
| Conversion : | $\mathcal{S}_A$ | $\mathcal{X}$ | $\mathcal{S}_B$ |
| | $37\mathbb{G} + 4P$ | $47\mathbb{G} + 2\tilde{\mathbb{G}} + 4P$ | $13\mathbb{G}$ |
| User Audit : | $\mathcal{U}_i$ | *with c denoting the amount* | |
| | $3c \cdot \mathbb{G}$ | *of conversions for* $\mathcal{U}_i$ | |

As a benchmark, using the BN254 curve of the Apache Milagro Cryptographic Library (AMCL, C-Version), exponentiations in $\mathbb{G}$, $\tilde{\mathbb{G}}$, and $\mathbb{G}_t$ require 0.6ms, 1.0ms, and 1.4ms respectively, and a pairing costs 1.6ms (executed on an Intel i5-4300U CPU). Thus, in terms of computational time the conversion takes 28.6ms for $\mathcal{S}_A$, 36.6ms for $\mathcal{X}$, and 7.8ms for $\mathcal{S}_B$ which we believe to be reasonably efficient for practical use.

## 7. Conclusion & Open Problems

In this paper we have shown how to construct a pseudonym system with build-in user auditability that does not require a fully trusted auditor or converter. Our pseudonym system overcomes the seemingly conflict of providing user-specific audits by an oblivious converter, and positively answers the open question from [7]. The converter generates and converts (un)linkable pseudonyms in a blind way, and also generates audit logs for each conversion request that allow the concerned user, and only him, to learn about the processing of his pseudonyms. The computational complexity for the user part is rather minimal, and in particular independent of the number of users in the pseudonym system. Further, our scheme provides better privacy for pseudonym generation than [7], as the converter no longer learns which user wishes to generate a pseudonym towards which server, that was inherently revealed in the scheme of Camenisch and Lehmann. Our construction makes use of a number of new building blocks and a tag-chaining approach which we believe to be of independent interest.

An interesting open question is how to tolerate fully malicious behaviour of the converter, e.g., via distributing the logging procedure or using recent advances in the block chain area, ideally without imposing a significant performance loss. Another area for future work is to further enhance the power of the user, such that he can decide which pseudonym conversions are allowed while remaining anonymous to the converter.

**Acknowledgements.** We thank Manu Drijvers for providing the benchmark measurements. This work was supported by the European Commission through the Seventh Framework Programme under grant agreement #321310 for the PERCY grant.

## References

[1] Michel Abdalla, Mihir Bellare, and Gregory Neven. Robust encryption. In Daniele Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 480–497. Springer, Heidelberg, February 2010.

[2] Giuseppe Ateniese, Jan Camenisch, Susan Hohenberger, and Breno de Medeiros. Practical group signatures without random oracles. Cryptology ePrint Archive, Report 2005/385, 2005. http://eprint.iacr.org/2005/385.

[3] Boaz Barak, Ran Canetti, Yehuda Lindell, Rafael Pass, and Tal Rabin. Secure computation without authentication. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 361–377. Springer, Heidelberg, August 2005.

[4] Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. Randomizable proofs and delegatable anonymous credentials. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 108–125. Springer, Heidelberg, August 2009.

[5] Dan Boneh and Xavier Boyen. Efficient selective-ID secure identity based encryption without random oracles. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 223–238. Springer, Heidelberg, May 2004.

[6] Jan Camenisch, Aggelos Kiayias, and Moti Yung. On the portability of generalized Schnorr proofs. In Antoine Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 425–442. Springer, Heidelberg, April 2009.

[7] Jan Camenisch and Anja Lehmann. (Un)linkable pseudonyms for governmental databases. In Indrajit Ray, Ninghui Li, and Christopher Kruegel:, editors, *ACM CCS 15*, pages 1467–1479. ACM Press, October 2015.

[8] Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 126–144. Springer, Heidelberg, August 2003.

[9] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups (extended abstract). In Burton S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 410–424. Springer, Heidelberg, August 1997.

[10] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000. http://eprint.iacr.org/2000/067.

[11] Austrian Citizen Card. http://www.a-sit.at/de/dokumente_publikationen/flyer/buergerkarte_en.php.

[12] Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In Serge Vaudenay, editor, *PKC 2005*, volume 3386 of *LNCS*, pages 416–431. Springer, Heidelberg, January 2005.

[13] Matthias Fitzi, Martin Hirt, and Ueli M. Maurer. Trading correctness for privacy in unconditional multi-party computation (extended abstract). In Hugo Krawczyk, editor, *CRYPTO'98*, volume 1462 of *LNCS*, pages 121–136. Springer, Heidelberg, August 1998.

[14] David Galindo and Eric R Verheul. Microdata sharing via pseudonymizatio. Joint UNECE/Eurostat work session on statistical data confidentiality, 2007.

[15] Jens Groth. Efficient fully structure-preserving signatures for large messages. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 239–259. Springer, Heidelberg, November / December 2015.

[16] Martin Hirt, Christoph Lucas, and Ueli Maurer. A dynamic tradeoff between active and passive corruptions in secure multi-party computation. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 203–219. Springer, Heidelberg, August 2013.

[17] Stanislaw Jarecki, Aggelos Kiayias, Hugo Krawczyk, and Jiayu Xu. Highly-efficient and composable password-protected secret sharing (or: How to protect your bitcoin wallet online). In *IEEE European Symposium on Security and Privacy, EuroS&P 2016*, pages 276–291. IEEE, 2016.

[18] Stanislaw Jarecki and Xiaomin Liu. Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 577–594. Springer, Heidelberg, March 2009.

[19] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer, Heidelberg, May 1999.

[20] Tobias Pulls, Roel Peeters, and Karel Wouters. Distributed privacy-preserving transparency logging. In *Proceedings of the 12th annual ACM Workshop on Privacy in the Electronic Society, WPES 2013*, pages 83–94. ACM, 2013.

[21] Bruce Schneier and John Kelsey. Cryptographic support for secure logs on untrusted machines. In *Proceedings of the 7th USENIX Security Symposium*. USENIX Association, 1998.

[22] My Number (Japanese Social Security and Tax Number System). http://www.ny.us.emb-japan.go.jp/japaninfo/winter2016/03.html.

[23] Eric Verheul, Bart Jacobs, Carlo Meijer, Mireille Hildebrandt, and Joeri de Ruiter. Polymorphic encryption and pseudonymisation for personalised healthcare. Cryptology ePrint Archive, Report 2016/411, 2016. http://eprint.iacr.org/2016/411.

[24] Brent R. Waters, Edward W. Felten, and Amit Sahai. Receiver anonymity via incomparable public keys. In Sushil Jajodia, Vijayalakshmi Atluri, and Trent Jaeger, editors, *ACM CCS 03*, pages 112–121. ACM Press, October 2003.

[25] Douglas Wikström. A universally composable mix-net. In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 317–335. Springer, Heidelberg, February 2004.

[26] Karel Wouters, Koen Simoens, Danny Lathouwers, and Bart Preneel. Secure and privacy-friendly logging for egovernment services. In *Proceedings of the The Third International Conference on Availability, Reliability and Security, ARES 2008*, pages 1091–1096. IEEE Computer Society, 2008.

[27] Adam L. Young and Moti Yung. Semantically secure anonymity: Foundations of re-encryption. Cryptology ePrint Archive, Report 2016/341, 2016. http://eprint.iacr.org/2016/341.

[28] Vassilis Zikas, Sarah Hauser, and Ueli M. Maurer. Realistic failures in secure multi-party computation. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 274–293. Springer, Heidelberg, March 2009.