(Un)linkable Pseudonyms for Governmental Databases

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

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

ABSTRACT

When data maintained in a decentralized fashion needs to be synchronized or exchanged between different databases, related data sets usually get associated with a unique identifier. While this approach facilitates cross-domain data exchange, it also comes with inherent drawbacks in terms of controllability. As data records can easily be linked, no central authority can limit or control the information flow. Worse, when records contain sensitive personal data, as is for instance the case in national social security systems, such linkability poses a massive security and privacy threat. An alternative approach is to use domain-specific pseudonyms, where only a central authority knows the cross-domain relation between the pseudonyms. However, current solutions require the central authority to be a fully trusted party, as otherwise it can provide false conversions and exploit the data it learns from the requests. We propose an (un)linkable pseudonym system that overcomes those limitations, and enables controlled yet privacy-friendly exchange of distributed data. We prove our protocol secure in the UC framework and provide an efficient instantiation based on discrete-logarithm related assumptions.

Categories and Subject Descriptors

D.4.6 [**Security and Protection**]: Cryptographic control; H.2.7 [**Database Administration**]: Security, integrity, and protection; K.4.1 [**Public Policy Issues**]: Privacy

Keywords

pseudonyms; unique identifier; unlinkability; databases

1. INTRODUCTION

When data is collected and maintained on a large scale, that data often does not reside in a single database but is distributed over several databases and organisations, each being responsible for a particular aspect in the overall system. To still allow for collaborative operations and data exchange among the different entities, related data is then indexed with an identifier that is unique in the entire system.

An important example of such a distributed setting is a state-controlled social security system maintaining various sets of personal data. Therein users can interact with different entities, such as different health care providers, public and private pension funds, or tax authorities. All these entities can act autonomously in their respective domains and keep individual records for the users they interact with. In certain scenarios, the different entities also have to exchange data about particular users. For instance, assume a health care provider offers special discounts for people with low income and tax authorities store information about users' salaries. Then, to verify whether a user is eligible for a discount, the health care system together with the tax authority should be able to check if the user satisfies the criteria.

Global Identifiers.

Currently, the probably most prominent approach to enable such decentralized data management is to use unique global identifiers among all entities. In the context of social security systems, this is for instance implemented in the US, Sweden, and Belgium. Here, each citizen gets assigned a unique and nation-wide social security number. The advantage of this approach is that it naturally allows all entities within the system to correlate their individually maintained records. However, having such a unique identifier for each user is also a significant privacy threat: When data is lost or stolen, also any adversary obtaining the data can use the unique identifier to link all the different data sets together. Also, interactions of users with different entities become easily traceable.

Thus, the impact of security breaches is rather severe, which in turn, makes the data maintained by the individual entities a lucrative target for data thieves. In addition, as all entities can trivially link their records, the data exchange can hardly be controlled and authorized. However, in particular in the case of a social security system, a certain control to supervise and, if necessary, limit the data flow is usually desired. For instance in the Belgium system currently a central authority called "crossroads bank for social security" (CBSS) [13], serves as hub for all data exchange. Whenever social and private entities want to exchange data based on the global identification number, they have to request explicit authorization from the CBSS, as enforced by national law. From a privacy point of view, though, this added controllability makes the system even worse, as now a central authority learns which requests are made for which user. In a social security system, those requests can reveal quite sensi-

CCS'15, October 12–16, 2015, Denver, Colorado, USA.

ACM 978-1-4503-3832-5/15/10 ...$15.00.
DOI: http://dx.doi.org/10.1145/2810103.2813658 .

tive information itself. For instance, in the example outlined above, the central authority would learn from the requests which persons suffer from health issues and probably have low or no income, even if it has no access to the health and tax records itself. Also in terms of security it still assumes that all entities behave honestly and do not correlate their records without approval of the central authority.

Pseudonyms & Controlled Conversion.

Having such a central authority actually allows for a more privacy-friendly solution. Namely, a central authority (that we call *converter*) could derive and distribute entity-specific identifiers (aka *pseudonyms*), in a way that pseudonyms known by different entities can only be linked with the help of the converter. Thus, it would then even be technically enforced that different entities have to request permission from the converter, as without its help they would not be able to connect their records anymore. Of course, the latter argument only holds if the data sets maintained by the entities do not contain other unique identifying information which allows linkage without using the pseudonyms.

Such a pseudonymous identification system clearly improves the controllability of the data exchanges and also avoids imposing a unique identifier that makes the user traceable by default. Both are significant advantages compared with the solution where only a single global identifier is used throughout the entire system. However, as now the converter is indeed required in every request it yields a powerful entity that still must be trusted to not exploit the information it gathers.

Existing Solutions.

In existing solutions, the need to fully trust the converter seems in fact inherent. A similar pseudonymous framework using a central converter is for instance described by Galindo and Verheul [20]. Therein, the converter computes a pseudonym $nym_{i,A}$ based on main identifier uid_i and for server $\mathcal{S}_A$, as $nym_{i,A} = \mathsf{Enc}(k_A, uid_i)$, where Enc is a blockcipher and k_A a symmetric key that the converter has chosen for $\mathcal{S}_A$, but is only known to the converter. When an entity $\mathcal{S}_A$ then wishes to request some data for $nym_{i,A}$ from another entity $\mathcal{S}_B$, it sends the pseudonym to the converter. The converter then decrypts $nym_{i,A}$ to obtain uid_i and derives the local pseudonym $nym_{i,B}$ by computing $nym_{i,B} = \mathsf{Enc}(k_B, uid_i)$ for the key k_B it had chosen for $\mathcal{S}_B$. Thus, here the converter is necessarily modeled as a trusted third party, as it always learns the generated pseudonyms, the underlying uid_i and also has full control over the translations it provides (i.e., a corrupt converter could transform pseudonyms arbitrarily).

Another example is the Austrian eID system [12], which is one of the few eID solutions that allows one to derive entity-specific pseudonyms from the unique social security number. However, it currently only supports that unlinkable pseudonyms are created by the users themselves, but it does not consider a central authority that can provide a conversion service on a large scale. It is easy to imagine though, how such a converter could be realized. Roughly, a pseudonym $nym_{i,A}$ is computed as $\mathsf{H}(\mathsf{Enc}(k, uid_i)||\mathcal{S}_A)$, i.e., the encrypted main user identifier uid_i and the identifier of the respective entity $\mathcal{S}_A$ are concatenated and the hash value of both yields the pseudonym. Here, the key k is a global key that is used for all pseudonyms, but is again only known to the converter. In order to enable conversions between pseudonyms, the converter could simply keep a table with the related hash values and then perform the conversion based on looking up the corresponding value.

Hereby, the trust requirements for the converter can actually be reduced if one considers pseudonym generation and conversion as two different tasks. Then, only the entity responsible for pseudonym generation would have to know the key k under which the user identifiers are encrypted, whereas the converter merely keeps the hash table with the related pseudonyms. The converter would then only know which pseudonyms belong together, but can not determine for which particular user they are standing for. Thus, also during conversion, a malicious converter does not learn the particular user for which a conversion is requested anymore, but only his pseudonym.

However, the converter can still link all requests that are made for the same (unknown) user. As each query usually leaks some context information itself, being able to link all that information together might still allow the converter to fully identify the concrete user behind a pseudonym. For instance, regular queries for the same pseudonym to the pension fund might indicate that the person behind the pseudonym is older than 60 years, and queries to entities that are associated with a certain region such as local municipalities further reveal the place that person might live in.

Via the comparable CBSS authority in Belgium, several hundreds of million messages are exchanged every year, with a peak of 806 million messages in 2009. Using those values as a reference for the social security use case, one has to assume that a converter learning "only" the requests and their relation would still obtain a significant amount of context data. How context information and meta data can be leveraged to fully de-anonymize pseudonymized data sets, was recently impressively demonstrated for "anonymized" credit card transactions [15] and in the Netflix and AOL incidents [21, 5].

Thus, from a privacy and a security perspective it is clearly desirable to minimize the information a converter can collect as much as possible. This means, the converter should not even learn which requests relate to which pseudonyms.

Other Related Work.

There exists a line of work on reversible pseudonymization of data records, in particular in the eHealth context, aiming at de-sensitizing patient records [1, 22, 14, 8, 24]. The main focus in these works is to derive pseudonyms from unique patient identifiers, such that the pseudonyms do not reveal any information about the patient anymore, yet allow de-anonymization by a trusted party (or a combination of several semi-trusted parties). However, in all solutions, pseudonym generation must be repetitively unambiguous to preserve the correlation between all pseudonymized records. Consequently, data exchange is trivial and does not require a converter. Thus, pseudonyms are linkable by default, whereas our approach is the opposite: pseudonyms should be *unlinkable by default*, yet preserve the correlation which allows to re-establish the linkage only if necessary via a (potentially untrusted) converter.

Our Contribution.

In this paper we tackle the challenge of enabling privacy-friendly yet controlled data exchange in a decentralized system. That is, we propose an (un)linkable pseudonym system where a converter serves as central hub to ensure controllability. The converter establishes individual pseudonyms for each server derived from a unique main identifier that every user has, but without learning the derived pseudonyms. The converter is still the only authority that can link different pseudonyms together, but it does not learn the particular user or pseudonym for which such a translation is requested. The converter can not even tell if two data exchanges were done for the same pseudonym or for two different ones. Thus, the only information the converter still learns is that a server $\mathcal{S}_A$ wants to access data from a server $\mathcal{S}_B$. We consider this to be the right amount of information to balance control and privacy. For instance for the use case of a social security system, it might be allowed that the health care provider can request data from the tax authority but should not be able to access the criminal records of its registered users. Thus, there is no need to learn for which particular user a request is made, or whether several requests belong together. In our system, the converter is able to provide such access control but does not learn any additional information from the queries.

We start by formally defining the functional and security properties such an (un)linkable pseudonym system should ideally provide. Our definition is formulated in the universal composability framework, and thus comes with strong guarantees when composed with other UC secure protocols. We then describe our system using generic building blocks.

The idea of our solution to build pseudonyms by adding several layers of randomness to the user identifier, such that they allow for consistent (and blind) conversions yet hide the contained identifier towards the servers. Roughly, to generate a pseudonym $nym_{i,A}$ for a user uid_i on server $\mathcal{S}_A$, the converter first applies a verifiable PRF on uid_i and then raises the derived value to a secret exponent that it assigns for each server. The trick thereby is that those secret keys are known only to the converter, but not to the servers.

Now, consider the blind conversion procedure. It can of course be realized with a generic multiparty protocol, where the first server $\mathcal{S}_A$ inputs the pseudonym to be converted and the converter inputs all its secret keys, and the output of the second server $\mathcal{S}_B$ would be the converted pseudonym, provided that the input by $\mathcal{S}_A$ was a indeed a valid pseudonym. However, such a computation would be rather inefficient. We therefore aim to construct a specific protocol that achieves this efficiently.

We propose a blind conversion protocol that performs the conversion on *encrypted* pseudonyms, using a homomorphic encryption scheme. To transform a pseudonym from one server to another, the converter then exponentiates the encrypted pseudonym with the quotient of the secret keys of the two servers. The challenge is to make that entire process verifiable, ensuring that the conversion is done in a consistent way but without harming the privacy properties.

To ensure controllability in the sense that a server can only request conversions for pseudonyms it legitimately obtained via the converter, we also make use of a novel building block which we call *dual-mode signatures*. Those allow to obtain signatures on encrypted messages, which can then be "decrypted" to a signature on the underlying plaintext message. We also provide a concrete construction for those signatures based on the recent signature scheme by Abe et al. [2], which might be of independent interest. Our dual-mode signatures can be seen as a spezialised variant of commuting signatures [19], and therefore allow for more efficient schemes.

Finally, we prove that our protocol realizes our ideal functionality based on the security of the building blocks. We also provide concrete instantiations for all generic building blocks used in our construction which already come with optimizations and enhance the efficiency of our solution. When instantiated with the suggested primitives, our protocol is secure based on discrete-logarithm related assumptions.

2. SECURITY DEFINITION

In this section we first informally discuss the main entities and procedures in our (un)linkable pseudonym system and then define the desired security properties by describing how an ideal functionality would handle that task.

For the sake of simplicity, we will speak about *user* identifier uid_i, whenever we mean a unique identifier to which several distributed data sets should be related. However, it should not be misunderstood that our system is restricted to user data, as it can handle arbitrary related data sets distributed over several servers. The main entities in our system are a converter $\mathcal{X}$ and a set of servers $\mathbf{S} = \{\mathcal{S}_A, \mathcal{S}_B, \dots\}$.

The converter $\mathcal{X}$ is the central authority that blindly derives and converts the (un)linkable pseudonyms. More precisely, for a user identifier uid_i and server identifier $\mathcal{S}_A$, the converter can establish the server-specific pseudonym $nym_{i,A}$. However, this must be done in a way that only $\mathcal{S}_A$ is privy of the resulted $nym_{i,A}$.

All generated pseudonyms can also be verified by the servers. In particular, if a server $\mathcal{S}_A$ does know the underlying uid_i, and the converter allows for verification, it can check that $nym_{i,A}$ is indeed derived from uid_i. This is crucial to allow for a secure migration from an existing indexing system based on unique uid_i's to our pseudonymous system. However, such a verification must be explicitly allowed by the converter. Without his approval, a server even when knowing some uid_i could not verify whether it belongs to a certain pseudonym $nym_{i,A}$ or not.

A server $\mathcal{S}_A$ can then maintain data for some user uid_i who is known to him as $nym_{i,A}$. If $\mathcal{S}_A$ wants to access some data for the same underlying user from another server $\mathcal{S}_B$, it must initiate a conversion request via the converter. The converter is the only entity that can transform a pseudonym $nym_{i,A}$ into $nym_{i,B}$. However, $\mathcal{X}$ executes the conversion function in a blind manner, i.e., without learning $nym_{i,A}, nym_{i,B}$, the underlying uid_i or even if two requests are made for the same pseudonym or not. If a conversion is granted by $\mathcal{X}$, only $\mathcal{S}_B$ will learn the converted pseudonym $nym_{i,B}$. The subsequent data exchange between $\mathcal{S}_A$ and $\mathcal{S}_B$ can be handled using the query identifier qid that is used in the request and is mapped to $nym_{i,A}$ on $\mathcal{S}_A$'s and to $nym_{i,B}$ on $\mathcal{S}_B$'s domain.

Apart from all the privacy features it is of course crucial that pseudonyms are generated and converted in a *consistent* way. More precisely, the generated pseudonyms $nym_{i,A}$ must be unique for each server domain $\mathcal{S}_A$ and the conversion must be transitive and consistent with the pseudonym generation.

2.1 Ideal Functionality

We now formally define such an (un)linkable pseudonym system with blind conversion by describing an ideal functionality in the universal composability (UC) framework [11], which is a general framework for analyzing the security of cryptographic protocols. Roughly, a protocol is said to securely realize a certain ideal functionality $\mathcal{F}$, if an environment can not distinguish whether it is interacting with the real protocol or with $\mathcal{F}$ and a simulator. A protocol that is proven to be secure in the UC framework then enjoys strong security guarantees even under arbitrary composition with other (UC secure) protocols. At the end of the section we will also discuss how the aforementioned (informal) properties are enforced by our functionality.

In this paper, we assume static corruptions, meaning that 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 $sid = (sid', \mathcal{X}, \mathbf{S}, \mathbf{U}, \mathbf{N})$, for some converter and server identifiers $\mathcal{X}, \mathbf{S} = \{\mathcal{S}_A, \mathcal{S}_B, \dots\}$ and a unique string $sid' \in \{0,1\}^*$. Further, it must hold that $|\mathbf{N}| \geq |\mathbf{U}|$, where $\mathbf{U}$ denotes the space of user identifiers and $\mathbf{N}$ the pseudonym space. We also assume unique query identifiers $qid = (qid', \mathcal{S}_A, \mathcal{S}_B)$ for each conversion request, containing the identities of the communicating servers $\mathcal{S}_A$ and $\mathcal{S}_B$. Those unique session and query identifiers can be established, e.g., by exchanging random nonces between all involved parties and using the concatenation of all nonces as sid' and qid' respectively.

The definition of our ideal functionality $\mathcal{F}_{\mathsf{nym}}$ is presented in detail in Figure 1. For simplicity, we refer to $\mathcal{F}_{\mathsf{nym}}$ as $\mathcal{F}$ from now on. We also use the following writing conventions in order to reduce repetitive notation:

- At each invocation, $\mathcal{F}$ checks that sid has the form $sid = (sid', \mathcal{X}, \mathbf{S}, \mathbf{U}, \mathbf{N})$, with $|\mathbf{N}| \geq |\mathbf{U}|$. When we say that $\mathcal{F}$ receives input from or provides output to $\mathcal{S}_A$ or $\mathcal{X}$, we mean the particular $\mathcal{X}$ or $\mathcal{S}_A \in \mathbf{S}$ specified in the sid.
- For the CONVERT and PROCEED interfaces, $\mathcal{F}$ checks that $qid = (qid', \mathcal{S}_A, \mathcal{S}_B)$ and only considers the first message for each pair (sid, qid). Subsequent messages for the same (sid, qid) are ignored.
- When we say that $\mathcal{F}$ *outputs* a message to a party, this happens directly, i.e, the adversary neither sees the message nor can delay it.
- When we say that $\mathcal{F}$ sends a message m to $\mathcal{A}$ and waits for m' from $\mathcal{A}$, we mean that $\mathcal{F}$ chooses a unique execution identifier, saves the local variables and other relevant information for the current interface invocation, and sends m together with the identifier to $\mathcal{A}$. When $\mathcal{A}$ then invokes a dedicated resume interface with a message m' and an execution identifier, $\mathcal{F}$ looks up the information associated to the identifier and continues processing the request for input m'.
- When we say that $\mathcal{F}$ *proceeds only* under a certain condition, we implicitly assume that a failure message is sent to the caller whenever that condition is not fulfilled.

We now describe the behaviour of all interfaces also in a somewhat informal manner to clarify the security properties that our functionality provides.

Pseudonym Generation.

The NYMGEN interface allows a converter to trigger the generation of a pseudonym $nym_{i,A}$ for user uid_i and server $\mathcal{S}_A$. If no pseudonym for that combination of $uid_i, \mathcal{S}_A$ exists in $\mathcal{F}$, a new one is created. Thereby, if $\mathcal{X}$ or $\mathcal{S}_A$ are honest, the new pseudonym is chosen at random from $\mathbf{N}$. (In Figure 1 this is denoted by $nym_{i,A} \xleftarrow{\$} \mathbf{N}$.) Only if both, the converter and the server are corrupt, the adversary can provide the pseudonym $nym^*_{i,A}$. All generated pseudonyms are stored within $\mathcal{F}$ as $(\mathsf{nym}, sid, uid_i, \mathcal{S}_A, nym_{i,A})$, i.e, the records also include the underlying uid_i.

The generated pseudonym $nym_{i,A}$ is then output directly to $\mathcal{S}_A$. Thus, while the converter is the crucial entity to establish a server-specific pseudonym, it does not learn the pseudonym itself. The converter can additionally specify whether the server output shall consist solely of the pseudonym, or come in a verifiable manner. Verifiable means that the server $\mathcal{S}_A$ receives a pseudonym $nym_{i,A}$ together with an underlying uid_i, assuring that $nym_{i,A}$ indeed belongs to uid_i. Such verification is indicated with the flag $anon = 0$, whereas $anon = 1$ will hide the uid_i from $\mathcal{S}_A$. The reason to include the option $anon = 0$ and thus the "leakage" of uid_i is that a server might already know and use the uid_i and thus should be able to verify to which particular user a new pseudonym belongs to (and ideally delete the uid_i afterwards). Allowing this non-privacy-friendly option might appear counter-intuitive at a first glance. However, without having the possibility to verify whether a pseudonym indeed belongs to certain uid_i, the pseudonyms would have not much meaning. Thus, we consider the option $anon = 0$ crucial for bootstrapping such a system, but of course it should be used with care. We discuss further interesting strategies for pseudonym provisioning in Section 6.

When both $\mathcal{X}$ and $\mathcal{S}_A$ are corrupt, we also allow the generation of pseudonyms without assigning a proper $uid_i \in \mathbf{U}$ yet. Instead, the pseudonyms are stored for a "dummy" identifier $uid_i \notin \mathbf{U}$. However, such unassigned pseudonyms are only allowed as long as $\mathcal{X}$ does not wish to provide a *verifiable* pseudonym, i.e., where $anon = 1$.

Assign UID.

The ASSIGN interface is only available when the converter is corrupt. It allows the adversary to replace a "dummy" identifier $uid_i \notin \mathbf{U}$ in all records with a proper $uid'_i \in \mathbf{U}$, if uid'_i is not used in any other pseudonym record. After a pseudonym got a assigned a "proper" identifier uid'_i, the converter can now also distribute the pseudonyms for uid'_i in a verifiable manner via the NYMGEN interface.

This reflects that, as long as no honest server has verified the connection of a pseudonym to a particular user identifier, all $\mathcal{F}$ can guarantee is that pseudonyms that were derived from each other, all belong together (including transitive relations). However, the relation to a particular uid might still be unassigned. Only when the converter provides a *verifiable* pseudonym $nym_{i,A}$, i.e., it links a pseudonym to its underlying uid, this connection between $nym_{i,A}$ and uid_i becomes known, and must be guaranteed by the ideal functionality from then on. Which is exactly what this interface does.

Conversion Request.

The CONVERT interface allows a server $\mathcal{S}_A$ to initiate a conversion for some pseudonym $nym_{i,A}$ towards another server $\mathcal{S}_B$, and associated with query identifier qid. The request will only be processed if $nym_{i,A}$ is registered within $\mathcal{F}$. To ask for the converter's approval, $\mathcal{X}$ is then notified about the request. However, $\mathcal{X}$ only learns that $\mathcal{S}_A$ wants to run a conversion towards $\mathcal{S}_B$, but nothing else, in particular not the pseudonym $nym_{i,A}$ the request was initiated for.

1. `Pseudonym Generation.` On input of $(\mathsf{NYMGEN}, sid, uid_i, \mathcal{S}_A, anon)$ from converter $\mathcal{X}$:
   - If $\mathcal{X}$ is honest, only proceed if $uid_i \in \mathbf{U}$, where $\mathbf{U}$ is taken from sid.
   - If $\mathcal{X}$ is corrupt, also proceed if $uid_i \notin \mathbf{U}$, but only if $anon = 1$.
   - Send $(\mathsf{NYMGEN}, sid, \mathcal{S}_A)$ to $\mathcal{A}$ and wait for $(\mathsf{NYMGEN}, sid, \mathcal{S}_A, nym^*_{i,A})$ from $\mathcal{A}$.
   - If a pseudonym record $(\mathsf{nym}, sid, uid_i, \mathcal{S}_A, nym_{i,A})$ for $uid_i, \mathcal{S}_A$ exists, retrieve $nym_{i,A}$, otherwise create a new record where $nym_{i,A}$ is determined as follows:
     – if $\mathcal{X}$ or $\mathcal{S}_A$ are honest, set $nym_{i,A} \overset{\$}{\leftarrow} \mathbf{N}$,
     – if $\mathcal{X}$ and $\mathcal{S}_A$ are corrupt, and no other pseudonym record for $nym^*_{i,A}, \mathcal{S}_A$ exists, set $nym_{i,A} \leftarrow nym^*_{i,A}$. Abort otherwise.
   - If $anon = 1$, output $(\mathsf{NYMGEN}, sid, nym_{i,A}, \perp)$ to $\mathcal{S}_A$ and output $(\mathsf{NYMGEN}, sid, nym_{i,A}, uid_i)$ otherwise.
2. `Assign UID.` On input of $(\mathsf{ASSIGN}, sid, uid_i, uid'_i)$ from adversary $\mathcal{A}$:
   - Proceed only if $\mathcal{X}$ is corrupt, $uid_i \notin \mathbf{U}$ and $uid'_i \in \mathbf{U}$.
   - If no pseudonym record for uid'_i exists yet, replace the current "dummy" uid_i with the "real" uid'_i in all records $(\mathsf{nym}, sid, uid_i, \mathcal{S}_A, nym_{i,A})$, abort otherwise.
   - Send (ASSIGN, sid) to $\mathcal{A}$.
3. `Conversion Request.` On input of $(\mathsf{CONVERT}, sid, qid, nym_{i,A}, \mathcal{S}_B)$ from server $\mathcal{S}_A$:
   - Proceed only if a record $(\mathsf{nym}, sid, uid_i, \mathcal{S}_A, nym_{i,A})$ for $nym_{i,A}, \mathcal{S}_A$ exists.
   - Send $(\mathsf{CONVERT}, sid, qid)$ to $\mathcal{A}$ and wait for response $(\mathsf{CONVERT}, sid, qid)$ from $\mathcal{A}$.
   - Create a conversion record $(\mathsf{convert}, sid, qid, uid_i, \mathcal{S}_A, \mathcal{S}_B)$, where uid_i is taken from the pseudonym record for $nym_{i,A}, \mathcal{S}_A$.
   - Output $(\mathsf{CONVERT}, sid, qid, \mathcal{S}_A, \mathcal{S}_B)$ to $\mathcal{X}$.
4. `Conversion Response.` On input of $(\mathsf{PROCEED}, sid, qid)$ from converter $\mathcal{X}$:
   - Proceed only if a conversion record $(\mathsf{convert}, sid, qid, uid_i, \mathcal{S}_A, \mathcal{S}_B)$ for qid exists.
   - Send $(\mathsf{PROCEED}, sid, qid)$ to $\mathcal{A}$ and wait for $(\mathsf{PROCEED}, sid, qid, nym^*_{i,B})$ from $\mathcal{A}$.
   - If a pseudonym record $(\mathsf{nym}, sid, uid_i, \mathcal{S}_B, nym_{i,B})$ for $uid_i, \mathcal{S}_B$ exists, retrieve $nym_{i,B}$, otherwise create a new record where $nym_{i,B}$ is determined as follows:
     – if $\mathcal{X}$ or $\mathcal{S}_B$ are honest, set $nym_{i,B} \overset{\$}{\leftarrow} \mathbf{N}$,
     – if $\mathcal{X}$ and $\mathcal{S}_B$ are corrupt, and no other pseudonym record for $nym^*_{i,B}, \mathcal{S}_B$ exists, set $nym_{i,B} \leftarrow nym^*_{i,B}$. Abort otherwise.
   - Output $(\mathsf{CONVERTED}, sid, qid, \mathcal{S}_A, nym_{i,B})$ to $\mathcal{S}_B$.

Figure 1: Ideal Functionality $\mathcal{F}_{\mathsf{nym}}$ with $sid = (sid', \mathcal{X}, \mathbf{S}, \mathbf{U}, \mathbf{N})$

Conversion Response.

The PROCEED interface allows a converter to blindly complete a conversion request towards $\mathcal{S}_B$. The converted pseudonym $nym_{i,B}$ is either retrieved from an existing record using the internal knowledge of the underlying uid_i of the requested $nym_{i,A}$, or generated from scratch and stored together with uid_i in $\mathcal{F}$. Again, as long as not both, $\mathcal{X}$ and $\mathcal{S}_B$ are corrupt, the new pseudonym is a random value in $\mathbf{N}$. Finally, $\mathcal{S}_B$ (and only $\mathcal{S}_B$) receives the converted pseudonym $nym_{i,B}$. As $\mathcal{F}$ performs the conversion based on the underlying uid_i, the desired consistency properties are naturally guaranteed.

Discussion.

Overall, our ideal functionality defined in Figure 1 guarantees the following security and privacy properties even in the presence of corrupted entities.

Security against corrupt $\mathcal{S}_A, \mathcal{S}_B$: The pseudonyms received by the servers do not leak any information about the underlying user identifier uid_i, and can only be established via the converter. That is, even if a server $\mathcal{S}_A$ is corrupt and knows a user identifier uid_i, it can not predict the server-local pseudonym $nym_{i,A}$ himself. This is enforced by $\mathcal{F}$ as it generates pseudonyms only when requested or allowed (in a conversion) by $\mathcal{X}$ and produces pseudonyms that are merely random values in $\mathbf{N}$.

Further, for pseudonyms $nym_{i,A}$ and $nym_{i,B}$ held by two corrupt servers $\mathcal{S}_A$ and $\mathcal{S}_B$, the servers cannot tell – without the help of the converter – whether they belong to the same uid_i or not (of course only if the servers do not *both* know the underlying uid_i from a verifiable pseudonym as otherwise linkage is trivial). This follows again from the randomness of the pseudonyms. If only one server $\mathcal{S}_A$ or $\mathcal{S}_B$ is corrupt, then the corrupt server cannot use a conversion to learn any information about the corresponding pseudonym of the other honest server – even if the converter is corrupt too: our functionality does not give any output to $\mathcal{S}_A$, and $\mathcal{S}_B$ only receives $(qid, nym_{i,B})$, but not the initial $nym_{i,A}$.

Security against corrupt $\mathcal{X}$: If the converter is corrupt, it can trigger pseudonyms for uid_i's and servers $\mathcal{S}_A$ of its choice, however $\mathcal{X}$ can not determine or predict the pseudonym values whenever they are generated for an honest server (neither via pseudonym generation nor conversion). This is guaranteed by our definition as $\mathcal{F}$ generates new pseudonyms as random values in $\mathbf{N}$ and outputs them directly to the respective server, i.e, without the adversary seeing them.

Further, in a conversion request between two honest servers $\mathcal{S}_A$ and $\mathcal{S}_B$, a corrupt converter does not learn any information about the pseudonym $nym_{i,A}$ or $nym_{i,B}$, or even whether two request where made for the same pseudonym or not. This follows clearly from $\mathcal{F}$, as the only information $\mathcal{X}$ gets is that $\mathcal{S}_A$ requested a conversion towards $\mathcal{S}_B$. If the converter and one of the servers is corrupt, the adversary can of course learn the pseudonym of the corrupted server. If even both $\mathcal{S}_A, \mathcal{S}_B$ are corrupt, then the adversary obviously learns all involved pseudonyms, but this is unavoidable.

Our functionality also guarantees consistency, even in the presence of a corrupt converter. That is, even when generated or converted by a corrupt $\mathcal{X}$, honest servers are ensured that pseudonym generation is injective, conversion is transitive and both procedures generate consistent pseudonyms. This is naturally enforced by our functionality as $\mathcal{F}$ is aware of the underlying uid_i and uses that knowledge to ensure consistent conversions and a unique pseudonym for each $(uid_i, \mathcal{S}_A)$ combination.

3. BUILDING BLOCKS

Here, we introduce the building blocks for our construction. Apart from standard proof protocols, (verifiable) pseudorandom functions and homomorphic encryption we also need a new primitive which we call *dual-mode signatures.* We provide a formal definition for those signature schemes and also detail an instantiation based on the structure-preserving signature scheme by Abe et al. [2].

3.1 Bilinear Maps

Let $\mathbb{G}$, $\tilde{\mathbb{G}}$ and $\mathbb{G}_t$ be groups of prime order q. A map $e : \mathbb{G} \times \tilde{\mathbb{G}} \to \mathbb{G}_t$ must satisfy bilinearity, i.e., $e(g^x, \tilde{g}^y) = e(g, \tilde{g})^{xy}$; 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 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 $a \in \mathbb{G}$, $b \in \tilde{\mathbb{G}}$. If $\mathbb{G} = \tilde{\mathbb{G}}$ the map is symmetric and otherwise asymmetric.

3.2 Proof Protocols

When referring to the zero-knowledge proofs of knowledge of discrete logarithms and statements about them, we will follow the notation introduced by Camenisch and Stadler [10] and formally defined by Camenisch, Kiayias, and Yung [9].

For instance, $PK\{(a, b, c) : y = g^a h^b \wedge \tilde{y} = \tilde{g}^a \tilde{h}^c\}$ denotes a "*zero-knowledge Proof of Knowledge of integers a, b and c such that* $y = g^a h^b$ *and* $\tilde{y} = \tilde{g}^a \tilde{h}^c$ *holds,*" where $y, g, h, \tilde{y}, \tilde{g}$ and $\tilde{h}$ are elements of some groups $\mathbb{G} = \langle g \rangle = \langle h \rangle$ and $\tilde{\mathbb{G}} = \langle \tilde{g} \rangle = \langle \tilde{h} \rangle$. Given a protocol in this notation, it is straightforward to derive an actual protocol implementing the proof [9]. *SPK* denotes a signature proof of knowledge, that is a non-interactive transformation of a proof with the Fiat-Shamir heuristic [18].

Often we use a more abstract notation for proofs, e.g., by $\mathit{NIZK}\{(w) : \mathit{statement}(w)\}$ we denote any zero-knowledge proof protocol of knowledge of a witness w such that the $\mathit{statement}(w)$ is true. The idea is that when we use *SPK* we have the concrete realization in mind whereas with *NIZK* we mean any non-interactive zero-knowledge proof. 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.

3.3 (Verifiable) Pseudorandom Functions

To generate pseudonyms and verify their correct generation, we require a pseudorandom function PRF that allows for a proof that it was correctly computed. Informally, a pseudorandom function $\mathsf{PRF}(x, i)$ with key generation $(x, y) \xleftarrow{\$} \mathsf{PRFKGen}(1^\tau)$ is verifiable if it allows for an efficient proof that a value z is a proper PRF output for input i and secret key x: $\pi_z \xleftarrow{\$} \mathrm{NIZK}\{(x) :\ z = \mathsf{PRF}(x, i)\}(i, z)$.

Dodis and Yampolskiy [16] have proposed such a function, $\mathsf{PRF}_{\mathbb{G}}(x, i) = g^{1/(x+i)}$, which works in a cyclic group $\mathbb{G} = \langle g \rangle$ of order q. The pseudorandomness of which is based on the q-Decisional Diffie-Hellman Inversion problem [7]. The algorithms for it are as follows (here we deviate from their algorithms in the way we define the proof as we require that the proof algorithm be zero-knowledge).

The key generation $\mathsf{PRFKGen}_{\mathbb{G}}(1^\tau)$ generates a random secret key $x \in \mathbb{Z}_q$ with corresponding public key $y \leftarrow g^x$. The proof π_z of correct computation of the PRF, i.e., $z = \mathsf{PRF}_{\mathbb{G}}(\log_g y, i)$, does not need to be online extractable in our construction, and thus is as follows: $\pi_z \xleftarrow{\$} \mathrm{SPK}\{(x) :\ y = g^x \ \wedge\ g/z^i = z^x\}(y, g, i, z)$.

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

3.4 Homomorphic Encryption Schemes

We require an encryption scheme $(\mathsf{EncKGen}_{\mathbb{G}}, \mathsf{Enc}_{\mathbb{G}}, \mathsf{Dec}_{\mathbb{G}})$ that is semantically secure and that has a cyclic group $\mathbb{G}$ as message space. It consists of a key generation algorithm $(\mathit{epk}, \mathit{esk}) \xleftarrow{\$} \mathsf{EncKGen}_{\mathbb{G}}(1^\tau)$, where τ is a security parameter, an encryption algorithm $C \xleftarrow{\$} \mathsf{Enc}_{\mathbb{G}}(\mathit{epk}, m)$, with $m \in \mathbb{G}$, and a decryption algorithm $m \leftarrow \mathsf{Dec}_{\mathbb{G}}(\mathit{esk}, C)$. Sometimes we will make the randomness used in the encryption process explicit, in which case we will write $C \leftarrow \mathsf{Enc}_{\mathbb{G}}(\mathit{epk}, m, r)$, 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 $C_1 \in \mathsf{Enc}_{\mathbb{G}}(\mathit{epk}, m_1)$ and $C_2 \in \mathsf{Enc}_{\mathbb{G}}(\mathit{epk}, m_2)$, then $C_1 \odot C_2 \in \mathsf{Enc}_{\mathbb{G}}(\mathit{epk}, m_1 \cdot m_2)$. We will also use exponents to denote the repeated application of $\odot$, e.g., C^2 to denote $C \odot C$.

ElGamal Encryption (with a CRS Trapdoor).

We use the ElGamal encryption scheme, which is homomorphic and chosen plaintext secure. The semantic 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. τ, i.e., q is a τ-bit prime.

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

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

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

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 $\hat{y}$. For encryption, each ciphertext gets extended with an element $C_0 \leftarrow \hat{y}^r$, which will be ignored in normal decryption. In our security proof of the overall scheme, 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 Signatures Schemes

We require two different kinds of signature schemes: One signature scheme is needed for server $\mathcal{S}_A$ to sign a request to the converter so that later a server $\mathcal{S}_B$ can verify that what it gets from the converter stems indeed from server $\mathcal{S}_A$. For this, any standard signature scheme $(\mathsf{SigKGen}, \mathsf{Sign}, \mathsf{Vf})$ is sufficient. Such as scheme consists of a key generation algorithm $(spk, ssk) \overset{\$}{\leftarrow} \mathsf{SigKGen}(1^\tau)$, a signing algorithm $\sigma \overset{\$}{\leftarrow} \mathsf{Sign}(ssk, m)$, with $m \in \{0,1\}^*$, and a signature verification algorithm $\{0,1\} \leftarrow \mathsf{Vf}(spk, \sigma, m)$. The security definitions are standard and we thus do not repeat them here.

The second signature scheme we require is for the converter to sign pseudonyms. This scheme needs to support the signing of plain pseudonyms as well as encrypted pseudonyms. Also it needs to allow for (efficient) proofs of knowledge of a signature on a pseudonym that is encrypted. Commuting signatures [19] would fit our bill here. However, because of their generality, their use would make our construction much less efficient than what we present. The reason for that is that almost all inputs and outputs in the construction come with non-interactive proofs that they are well defined. As the definitions for commuting signatures also include these proofs, we cannot use (a subset of) these either. Blazy et al. [6] define signature schemes that can sign (randomizable) ciphertexts. Such schemes are a special case of commuting signatures and much closer to what we need. However, the security definition they give requires that the keys for the encryption scheme be honestly generated and the decryption key be available in the security game. This means that when using such a scheme in a construction, the decryption keys need to be extractable from adversarial parties and correct key generation enforced, which would lead to less efficient schemes. We therefore need to provide our own definition that does not suffer from the drawbacks discussed. We call this a dual-mode signature scheme as it allows one to sign messages in the plain as well as when they are contained in an encryption.

Finally, we point out that the dual-mode signatures are similar to blind signature schemes, where the signer also signs "encrypted" messages. Now the typical security definition for blind signatures requires only that an adversary be not able to produce more signatures than he ran signing protocols with the signer. That kind of definition would not be good enough for us – for our construction we need to be sure that the signer indeed only signs the message that is contained in the encryption. Further, the setting for which we will use those dual-mode signatures would not be realizable by blind signatures: in our protocol a server $\mathcal{S}_A$ encrypts a message under a public key of a server $\mathcal{S}_B$, the converter then signs a derivation of the ciphertext, and $\mathcal{S}_B$ finally decrypts the signature.

Dual-Mode Signature Schemes.

A *dual-mode* signature scheme consists of the algorithms $(\mathsf{SigKGen}_{\mathbb{G}}, \mathsf{Sign}_{\mathbb{G}}, \mathsf{EncSign}_{\mathbb{G}}, \mathsf{DecSign}_{\mathbb{G}}, \mathsf{Vf}_{\mathbb{G}}$ and also uses an encryption scheme $(\mathsf{EncKGen}_{\mathbb{G}}, \mathsf{Enc}_{\mathbb{G}}, \mathsf{Dec}_{\mathbb{G}})$ that has the group $\mathbb{G}$ as message space. In particular, the algorithms working with encrypted messages or signatures also get the keys $(epk, esk) \overset{\$}{\leftarrow} \mathsf{EncKGen}_{\mathbb{G}}(1^\tau)$ of the encryption scheme as input.

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

$\mathsf{Sign}_{\mathbb{G}}(ssk, m)$: On input a signing key ssk and a message $m \in \mathbb{G}$ outputs a signature σ.

$\mathsf{EncSign}_{\mathbb{G}}(ssk, epk, C)$: On input a signing key ssk, a public encryption key epk, and ciphertext $C = \mathsf{Enc}_{\mathbb{G}}(epk, m)$, outputs an "encrypted" signature $\overline{\sigma}$ of C.

$\mathsf{DecSign}_{\mathbb{G}}(esk, spk, \overline{\sigma})$: On input an "encrypted" signature $\overline{\sigma}$, secret decryption key esk and public verification key spk, outputs a standard signature σ.

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

For correctness, we require that for all $(spk, ssk) \overset{\$}{\leftarrow} \mathsf{SigKGen}_{\mathbb{G}}(1^\tau)$, all $(epk, esk) \overset{\$}{\leftarrow} \mathsf{EncKGen}_{\mathbb{G}}(1^\tau)$, all $m \in \mathbb{G}$, and all random choices in $\mathsf{Sign}(\cdot,\cdot)$, in $\mathsf{Enc}_{\mathbb{G}}(\cdot,\cdot)$, and $\mathsf{EncSign}_{\mathbb{G}}(\cdot,\cdot,\cdot)$, we have that $\mathsf{Vf}_{\mathbb{G}}(spk, \mathsf{Sign}_{\mathbb{G}}(ssk, m), m) = 1$ and $\mathsf{Vf}_{\mathbb{G}}(spk, \mathsf{DecSign}_{\mathbb{G}}(esk, spk, \mathsf{EncSign}_{\mathbb{G}}(ssk, epk, \mathsf{Enc}_{\mathbb{G}}(epk, m))), m) = 1$

In terms of security, we extend the standard unforgeability definition to allow the adversary to also get signatures on encrypted messages. Thereby, the oracle $\mathcal{O}_{\mathsf{EncSign}}$ will only sign correctly computed ciphertexts, which is modeled by providing an additional encryption oracle $\mathcal{O}_{\mathsf{Enc}}$ and only sign ciphertexts that were generated via $\mathcal{O}_{\mathsf{Enc}}$. When using the scheme, this can easily be enforced by asking the signature requester for a proof of correct ciphertext computation, and, indeed, in our construction such a proof is needed for other reasons as well. Note that we do not require that the "encrypted" signature output by $\mathsf{EncSign}_{\mathbb{G}}$ does not leak any information about the signature contained in it.

```
**Experiment** $\mathsf{Exp}^{\mathsf{DMSIG\text{-}forge}}_{\mathcal{A},\mathsf{DMSIG},\mathsf{Enc}_{\mathbb{G}}}(\mathbb{G}, \tau)$:
  $(spk, ssk) \overset{\$}{\leftarrow} \mathsf{SigKGen}(1^\tau)$
  $\mathbf{L} \leftarrow \emptyset$; $\mathbf{C} \leftarrow \emptyset$
  $(m^*, \sigma^*) \overset{\$}{\leftarrow} \mathcal{A}^{\mathcal{O}_{\mathsf{Sign}}(ssk,\cdot), \mathcal{O}_{\mathsf{Enc}}(\cdot,\cdot), \mathcal{O}_{\mathsf{EncSign}}(ssk,\cdot,\cdot)}(spk)$
    where $\mathcal{O}_{\mathsf{Sign}}$ on input $(m_i)$:
      add $m_i$ to the list of queried messages $\mathbf{L} \leftarrow \mathbf{L} \cup m_i$
      return $\sigma \overset{\$}{\leftarrow} \mathsf{Sign}_{\mathbb{G}}(ssk, m_i)$
    where $\mathcal{O}_{\mathsf{Enc}}$ on input $(epk_i, m_i)$:
      run $C_i \overset{\$}{\leftarrow} \mathsf{Enc}_{\mathbb{G}}(epk_i, m_i)$ and add $(epk_i, C_i, m_i)$ to $\mathbf{C}$
      return $C_i$
    where $\mathcal{O}_{\mathsf{EncSign}}$ on input $(epk_i, C_i)$:
      retrieve $(epk_i, C_i, m_i)$ from $\mathbf{C}$, abort if it doesn't exist;
      add $m_i$ to the list of queried messages $\mathbf{L} \leftarrow \mathbf{L} \cup m_i$
      return $\overline{\sigma} \overset{\$}{\leftarrow} \mathsf{EncSign}_{\mathbb{G}}(ssk, epk_i, C_i)$
  return 1 if $\mathsf{Vf}_{\mathbb{G}}(spk, \sigma^*, m^*) = 1$ and $m^* \notin \mathbf{L}$
```

Figure 2: Unforgeability experiment for dual-mode signatures

Definition 3.1. (Unforgeability of Dual-Mode Signatures). *We say a dual-mode signature scheme is* unforgeable *if for any efficient algorithm $\mathcal{A}$ the probability that the experiment given in Figure 2 returns 1 is negligible (as a function of τ).*

AGOT+ (Dual-Mode) Signature Scheme.

To instantiate the building block of dual-mode signatures we will use an extension of the structure-preserving signature scheme by Abe et al. [2], which we denote as AGOT+ scheme. First, we recall the original AGOT scheme $(\mathsf{SigKGen}_{\mathbb{G}}, \mathsf{Sign}_{\mathbb{G}}, \mathsf{Vf}_{G})$ slightly adapted to our notation, and then we describe how to instantiate the additional algorithms $\mathsf{EncSign}_{\mathbb{G}}$ and $\mathsf{DecSign}_{\mathbb{G}}$ with respect to a homomorphic encryption scheme $(\mathsf{EncKGen}_{\mathbb{G}}, \mathsf{Enc}_{\mathbb{G}}, \mathsf{Dec}_{\mathbb{G}})$.

AGOT assumes the availability of system parameters $crs = (q, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_t, e, g, \tilde{g}, x)$ consisting of $(q, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_t, e, g, \tilde{g}) \xleftarrow{\$} \mathcal{G}(1^\tau)$ and an additional random group element $x \xleftarrow{\$} \mathbb{G}$, that is, the key generation is split in two parts, one that generates the public parameters and one that generates the public and secret keys for the signer. For our construction, the former part will also generate the group $\mathbb{G}$ that will also be the message space of the encryption scheme. Thus, $\mathsf{SigKGen}_\mathbb{G}$ becomes that second part of the AGOT key generation, abusing notation, we give it the public parameters as input instead of the security parameter τ. For all other algorithms, we assume that the public parameters, in particular the group $\mathbb{G}$, are given as implicit input.

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

$\mathsf{Sign}_\mathbb{G}(ssk, m)$: On input a message $m \in \mathbb{G}$ and key $ssk = v$, choose a random $u \xleftarrow{\$} \mathbb{Z}_q^*$, and output the signature $\sigma = (r, s, t, w)$ including the randomization token w where:

$$r \leftarrow \tilde{g}^u, \quad s \leftarrow (m^v \cdot x)^{1/u}, \quad t \leftarrow (s^v \cdot g)^{1/u}, \quad w \leftarrow g^{1/u}.$$

$\mathsf{Vf}_\mathbb{G}(spk, \sigma, m)$: Parse $\sigma = (r, s, t, w')$ and $spk = y$ and accept if and only if $m, s, t \in \mathbb{G}, r \in \tilde{\mathbb{G}}$, and

$$e(s, r) = e(m, y) \cdot e(x, \tilde{g}), \quad e(t, r) = e(s, y) \cdot e(g, \tilde{g}).$$

Note that for notational simplicity, we consider w part of the signature, i.e., $\sigma = (r, s, t, w)$, but that the verification equation does not perform any check on w. As pointed out by Abe et al., a signature $\sigma = (r, s, t)$ can be randomized using the randomization token w to obtain a signature $\sigma' = (r', s', t')$ by picking a random $u' \xleftarrow{\$} \mathbb{Z}_q^*$ and computing

$$r' \leftarrow r^{u'}, \quad s' \leftarrow s^{1/u'}, \quad t' \leftarrow (tw^{(u'-1)})^{1/u'^2}.$$

This randomization feature is useful to efficiently prove knowledge of a signature on an encrypted message, which is needed in our protocol. We show in Section 5 how such a proof can be constructed.

Now, we present the additional algorithms that allow to obtain signatures on encrypted messages M[1].

$\mathsf{EncSign}_\mathbb{G}(ssk, epk, M)$: On input a proper encryption $M = \mathsf{Enc}_\mathbb{G}(epk, m)$ of a message $m \in \mathbb{G}$ under epk, and secret key$ssk = v$, choose a random $u \xleftarrow{\$} \mathbb{Z}_q^*$, and output the (partially) encrypted signature $\bar{\sigma} = (r, S, T, w)$:

$$r \leftarrow \tilde{g}^u, \quad S \leftarrow (M^v \odot \mathsf{Enc}_\mathbb{G}(epk, x))^{1/u},$$
$$T \leftarrow (S^v \odot \mathsf{Enc}_\mathbb{G}(epk, g))^{1/u}, \quad w \leftarrow g^{1/u}.$$

$\mathsf{DecSign}_\mathbb{G}(esk, spk, \overline{\sigma})$: Parse $\overline{\sigma} = (r, S, T, w)$, compute $s \leftarrow \mathsf{Dec}_\mathbb{G}(esk, S)$, $t \leftarrow \mathsf{Dec}_\mathbb{G}(esk, T)$ and output $\sigma = (r, s, t, w)$.

It is not hard to see that $\sigma = (r, s \leftarrow \mathsf{Dec}_\mathbb{G}(esk, S), t \leftarrow \mathsf{Dec}_\mathbb{G}(esk, T), w)$ is a valid signature on $m \leftarrow \mathsf{Dec}_\mathbb{G}(esk, M)$, and that the distribution of these values is the same as when m was signed directly. More formally, we prove that the AGOT scheme extended with the above algorithms $\mathsf{EncSign}_\mathbb{G}, \mathsf{DecSign}_\mathbb{G}$ yields an unforgeable dual-mode signature scheme. The proof is given in the full paper.

Theorem 3.2 (Unforgeability of AGOT+). *If the AGOT signature scheme* $(\mathsf{SigKGen}_\mathbb{G}, \mathsf{Sign}_\mathbb{G}, \mathsf{Vf}_\mathbb{G})$ *is an unforgeable signature scheme then, together with the algorithms* $\mathsf{EncSign}_\mathbb{G}, \mathsf{DecSign}_\mathbb{G}$ *described above, the AGOT+ scheme* $(\mathsf{SigKGen}_\mathbb{G}, \mathsf{Sign}_\mathbb{G}, \mathsf{EncSign}_\mathbb{G}, \mathsf{DecSign}_\mathbb{G}, \mathsf{Vf}_G)$ *is an unforgeable dual-mode signature scheme.*

For our construction, we also require the signer to prove that it computed the signature on an encrypted message correctly. In Section 5 we describe how such a proof can be done. (Intuitively, one would think that one could just decrypt and then verify whether the result is a valid signature. However, we cannot do this in the security proof of our pseudonym scheme where we reduce to the security of the homomorphic encryption scheme, as then we don't have a decryption oracle.)

4. OUR PROTOCOL

In this section we present our protocols for an (un)linkable pseudonym system. We first give a high-level idea and then explain the detailed construction.

Roughly, the computation of pseudonyms is done in several layers, each adding randomness to the process such that the final pseudonym $nym_{i,A}$ is indistinguishable from a random value (if not both, $\mathcal{X}$ and $\mathcal{S}_A$, are corrupt) as required by our ideal functionality. At the same time, the pseudonyms must still have some (hidden) structure, which allows the consistent transformation of pseudonyms by the converter.

The main idea is to let the converter first derive a pseudorandom "core identifier" $z_i \leftarrow \mathsf{PRF}_\mathbb{G}(x_\mathcal{X}, uid_i)$ from uid_i and for secret key $x_\mathcal{X}$. From the unique core identifier z_i, the converter then derives its pseudonym contribution using a *secret* exponent x_A that it chooses for each server $\mathcal{S}_A \in \mathbf{S}$, but never reveals to them.

For the blind conversion, we use homomorphic encryption so that the first server $\mathcal{S}_A$ can encrypt the pseudonym for the second server $\mathcal{S}_B$ hand this encryption to the converter, who, using the homomorphic properties of the encryption scheme, raises the encrypted pseudonym to the quotient of the two servers' secret keys, thereby transforming the encrypted pseudonym.

The tricky part is to make this whole pseudonym generation and conversion process verifiable and consistent, but without harming the unlinkability and blindness properties. In particular, for pseudonym generation a server $\mathcal{S}_A$ must be ensured that it receives correctly formed pseudonyms. For conversion, $\mathcal{S}_A$ needs to prove to the converter that it encrypted a valid pseudonym, and the converter needs to prove to the server $\mathcal{S}_B$ that it applied the conversion correctly. This is achieved by a careful composition of nested encryption, dual-mode signatures which allow signing of plain and encrypted messages, and zero-knowledge proofs.

In the following we give the detailed description of our protocol and also provide some intuition for the protocol design.

4.1 Detailed Description

We now describe our protocol assuming that a certificate authority functionality $\mathcal{F}_\text{CA}$, a secure message transmission functionality $\mathcal{F}_\text{SMT}$ (enabling authenticated and encrypted communication), and a common reference string functionality $\mathcal{F}_\text{CRS}$ are available to all parties. For details of those

[1] In the AGOT+ scheme, we write M to denote the encryption of a message m, instead of C. Likewise, capital letters S, T denote the encrypted versions of the values s, t that would be computed in a standard AGOT signature.

functionalities we refer to [11]. $\mathcal{F}_{\text{CRS}}$ provides all parties with the system parameters, consisting of the security parameter τ and a cyclic group $\mathbb{G} = \langle g \rangle$ of order q (which is a τ-bit prime). In the description of the protocol, we assume that parties call $\mathcal{F}_{\text{CA}}$ to retrieve the necessary key material whenever they use a public key of another party. Further, if any of the checks in the protocol fails, the protocol ends with a failure message.

```
Converter Setup:
  $(epk_{\mathcal{X}}, esk_{\mathcal{X}}) \xleftarrow{\$} \mathsf{EncKGen}_{\mathbb{G}}(1^\tau)$
  $(x_{\mathcal{X}}, y_{\mathcal{X}}) \xleftarrow{\$} \mathsf{PRFKGen}_{\mathbb{G}}(1^\tau)$
  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 $x_A \xleftarrow{\$} \mathbb{Z}_q$ and compute $y_A \leftarrow g^{x_A}$
  store $sk_{\mathcal{X}} \leftarrow (esk_{\mathcal{X}}, x_{\mathcal{X}}, \{x_A, ssk_{\mathcal{X},A}\}_{\forall \mathcal{S}_A \in \mathbf{S}})$
  register $pk_{\mathcal{X}} \leftarrow (epk_{\mathcal{X}}, y_{\mathcal{X}}, \{y_A, spk_{\mathcal{X},A}\}_{\forall \mathcal{S}_A \in \mathbf{S}})$ with $\mathcal{F}_{\text{CA}}$

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

Figure 3: Setup of Converter and Servers

Setup.

Before starting a new instance of our (un)linkable pseudonym system, we assume that the converter and all servers use standard techniques [11, 4] to agree on a session identifier $sid = (sid', \mathcal{X}, \mathbf{S}, \mathbf{U}, \mathbf{N})$ where sid' is a fresh and unique string, $\mathcal{X}$ and $\mathbf{S} = \{\mathcal{S}_A, \mathcal{S}_B, \dots\}$ denote the identities of the communicating parties, and $\mathbf{U} = \mathbb{Z}_q$ and $\mathbf{N} = \mathbb{G}$ define the domain of user identifiers and pseudonyms respectively. Then, whenever a new sid has been agreed on, all specified entities $\mathcal{X}$ and $\mathbf{S}$ generate their keys as described in Figure 3. For simplicity, we assume that the converter setup is trusted and discuss in the full paper how this assumption can be relaxed (e.g., by adding NIZKs proving knowledge of the secret keys, or distributed key generation).

Pseudonym Generation.

A pseudonym $nym_{i,A}$ for main identifier uid_i and server $\mathcal{S}_A$ is jointly computed by the server and the converter $\mathcal{X}$, as depicted in Figure 4. The generation is initiated by the converter and starts by applying a pseudorandom function to uid_i obtaining a secret "core identifier" $z_i \leftarrow \mathsf{PRF}_{\mathbb{G}}(x_{\mathcal{X}}, uid_i)$. As $x_{\mathcal{X}}$ is a secret key known only to the converter, the servers are not privy of the mapping between uid_i and z_i. From the core identifier z_i – which is the same for all servers in $\mathbf{S}$ – the converter then derives a server-specific "inner pseudonym" $xnym_{i,A} \leftarrow z_i^{x_A}$ for a secret conversion value x_A that $\mathcal{X}$ chooses internally for every server $\mathcal{S}_A$, but never reveals to them. By using a verifiable PRF and proving correctness of the computation in π_{nym}, the entire process of deriving the inner pseudonym $xnym_{i,A}$ can be verified by the server. If $anon = 0$, i.e., the pseudonym should be verifiably derived from a particular uid_i that is also given to the server, the proof is done w.r.t. that uid_i, whereas for $anon = 1$, π_{nym} only shows that the pseudonym was formed correctly for *some* uid_i. In the latter case, the proof actually shows that the pseudonym is of the correct form $xnym_{i,A} = z_i^{x_A}$ for some $z_i \in \mathbb{G}$ and also allows for extraction of z_i as this will be required in the security proof.

The inner pseudonym $xnym_{i,A}$ gets also accompanied with a server-specific signature σ_{nym} generated by the converter (using a dedicated signing key for each server). This signature will be crucial in a conversion request to ensure that only the server $\mathcal{S}_A$, for which the pseudonym was intended for, can subsequently use it in a conversion. We use the dual-mode signature for that purpose, as the converter needs to sign pseudonyms also in a blind way when they are generated via a conversion request.

When receiving a correctly signed and derived $xnym_{i,A}$, the server $\mathcal{S}_A$ then adds the final pseudonym layer by applying a pseudorandom permutation to $xnym_{i,A}$ for secret key k_A as $nym_{i,A} \leftarrow \mathsf{PRP}_{\mathbb{G}}(k_A, xnym_{i,A})$. This ensures that the server's output $nym_{i,A}$ cannot be linked to $xnym_{i,A}$ or uid_i by a corrupt converter.

Conversion Request.

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 $\mathcal{X}$, as described in Figure 5. Each request also comes with a unique query identifier qid (which can be established through the same standard techniques as sid). To achieve blindness of the request towards $\mathcal{X}$, the server encrypts the unwrapped inner pseudonym $xnym_{i,A}$ under $\mathcal{S}_B$'s public key. We also add a second layer of encryption using $\mathcal{X}$'s public key. This nested encryption is necessary to allow $\mathcal{X}$ to later prove correctness of a conversion towards the target server $\mathcal{S}_B$, but without $\mathcal{S}_B$ learning the value $xnym_{i,A}$. The signature σ_C of $\mathcal{S}_A$ on the nested encryption serves the same purpose. Both, the signature and proof are thereby bound to the query identifier qid, such that a corrupt $\mathcal{X}$ cannot reuse the values in a different session.

Finally, we also want to ensure that $\mathcal{S}_A$ can only trigger conversions of correct pseudonyms that "belong" to the server. Therefore, $\mathcal{S}_A$ has to prove in π_A that the ciphertext sent in the request contains a pseudonym $xnym_{i,A}$ that is signed under the correct key of the converter (but without revealing the signature).

When the converter $\mathcal{X}$ receives such a request, it first verifies the signature σ_C and proof π_A. If both are valid, $\mathcal{X}$ asks the environment whether it shall proceed. This is the hook to some external procedure which decides if the conversion from $\mathcal{S}_A$ to $\mathcal{S}_B$ shall be granted or not.

Conversion Response.

If the converter gets the approval to proceed, $\mathcal{X}$ and $\mathcal{S}_B$ complete the conversion as depicted in Figure 5. First, $\mathcal{X}$ uses the homomorphic property of the encryption scheme and raises the encrypted pseudonym to the quotient x_B/x_A of the two servers' secret conversion keys, thereby blindly transforming the encrypted inner pseudonym into $xnym_{i,B}$. The converter also re-randomizes the ciphertext by multiplying an encryption of "1" which is crucial for proofing unlinkability. To allow $\mathcal{S}_B$ to subsequently use the obtained pseudonym, the converter also "blindly" signs the encrypted $xnym_{i,B}$ with the dual-mode signature scheme. For ensuring consistency of a conversion (even in the presence of a corrupt converter), $\mathcal{X}$ proves correctness of the transformation in $\pi_{\mathcal{X}}$. The converter then sends the encrypted inner pseudonym $xnym_{i,B}$ as C'' with encrypted signature $\overline{\sigma}_{nym}$ to $\mathcal{S}_B$, and also forwards the received tuple (C, σ_C, π_A).

Step1. Upon input $(\mathsf{NYMGEN}, sid, uid_i, \mathcal{S}_A, anon)$**, converter** $\mathcal{X}$ **generates its pseudonym contribution:**

a) Check that $uid_i \in \mathbb{Z}_q$, and if so compute $xnym_{i,A} \leftarrow \mathsf{PRF}_{\mathbb{G}}(x_{\mathcal{X}}, uid_i)^{x_A}$ and $\sigma_{nym} \leftarrow \mathsf{Sign}_{\mathbb{G}}(ssk_{\mathcal{X},\mathcal{A}}, xnym_{i,A})$.

b) Prove correctness of the pseudonym generation in the proof π_{nym}:

$$\text{if } anon = 0: \quad \pi_{nym} \overset{\$}{\leftarrow} \text{NIZK}\{(x_A, x_{\mathcal{X}}, z_i): \ xnym_{i,A} = z_i^{x_A} \ \wedge \ y_A = g^{x_A} \ \wedge \ z_i = \mathsf{PRF}_{\mathbb{G}}(x_{\mathcal{X}}, uid_i)\}(sid) \ .$$

$$\text{if } anon = 1: \quad \pi_{nym} \overset{\$}{\leftarrow} \text{NIZK}\{(x_A, \underline{z_i}): \ xnym_{i,A} = z_i^{x_A} \ \wedge \ y_A = g^{x_A} \ \wedge \ z_i \in \mathbb{G}\}(sid), \quad \text{and set } uid_i \leftarrow \perp \ .$$

c) Send $(sid, xnym_{i,A}, \sigma_{nym}, \pi_{nym}, uid_i)$ via $\mathcal{F}_{\text{SMT}}$ to $\mathcal{S}_A$, end with no output.

Step2. Upon receiving $(sid, xnym_{i,A}, \sigma_{nym}, \pi_{nym}, uid_i)$ **from** $\mathcal{X}$**,** S_A **verifies input and derives final pseudonym:**

a) Verify that $\mathsf{Vf}_{\mathbb{G}}(spk_{\mathcal{X},A}, \sigma_{nym}, xnym_{i,A}) = 1$ and that the proof π_{nym} is correct w.r.t. y_A and $y_{\mathcal{X}}, uid_i$ (if $uid_i \neq \perp$).

b) Compute $nym_{i,A} \leftarrow \mathsf{PRP}_{\mathbb{G}}(k_A, xnym_{i,A})$, store $(sid, nym_{i,A}, \sigma_{nym})$ and end with output $(\mathsf{NYMGEN}, sid, nym_{i,A}, uid_i)$.

Figure 4: Pseudonym Generation

When $\mathcal{S}_B$ receives a conversion request, it first checks that σ_C, π_A are valid, ensuring that the request indeed was triggered by $\mathcal{S}_A$ for query qid and for the pseudonym contained in C. When $\mathcal{S}_B$ also verified the correctness of the conversion via $\pi_{\mathcal{X}}$, it decrypts $xnym_{i,B}$ and corresponding signature σ_{nym}. The final pseudonym $nym_{i,B}$ is again derived using the $\mathsf{PRP}_{\mathbb{G}}$. It stores the pseudonym and signature, and outputs $nym_{i,B}$ together with the query identifier qid.

4.2 Security and Efficiency

Our protocol described above securely realizes the ideal functionality $\mathcal{F}_{\mathsf{nym}}$ defined in Section 2. The detailed proof of the following theorem is given in the full paper.

THEOREM 4.1. *The (un)linkable pseudonym system described in Section 4 securely implements the ideal functionality* $\mathcal{F}_{\mathsf{nym}}$ *defined in Section 2 in the* $(\mathcal{F}_{CA}, \mathcal{F}_{CRS}, \mathcal{F}_{SMT})$ *hybrid-model, provided that*

- $(\mathsf{EncKGen}_{\mathbb{G}}, \mathsf{Enc}_{\mathbb{G}}, \mathsf{Dec}_{\mathbb{G}})$ *is a semantically secure homomorphic encryption scheme,*
- $(\mathsf{SigKGen}_{\mathbb{G}}, \mathsf{Sign}_{\mathbb{G}}, \mathsf{EncSign}_{\mathbb{G}}, \mathsf{DecSign}_{\mathbb{G}}, \mathsf{Vf}_{\mathbb{G}})$ *is an unforgeable dual-mode signature scheme (as defined in Def. 3.1),*
- $(\mathsf{SigKGen}, \mathsf{Sign}, \mathsf{Vf})$ *is an unforgeable signature scheme,*
- $(\mathsf{PRFKGen}_{\mathbb{G}}, \mathsf{PRF}_{\mathbb{G}})$ *is a secure and verifiable pseudorandom function,*
- $(\mathsf{PRPKGen}_{\mathbb{G}}, \mathsf{PRP}_{\mathbb{G}})$ *is a secure pseudorandom permutation,*
- *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* $\mathbb{G}$.

When instantiated with the ElGamal encryption scheme for $(\mathsf{EncKGen}_{\mathbb{G}}, \mathsf{Enc}_{\mathbb{G}}, \mathsf{Dec}_{\mathbb{G}})$, with Schnorr signatures [25, 23] for $(\mathsf{SigKGen}, \mathsf{Sign}, \mathsf{Vf})$, with the AGOT+ dual-mode signature scheme for $(\mathsf{SigKGen}_{\mathbb{G}}, \mathsf{Sign}_{\mathbb{G}}, \mathsf{EncSign}_{\mathbb{G}}, \mathsf{DecSign}_{\mathbb{G}}, \mathsf{Vf}_{\mathbb{G}})$, with the Dodis-Yampolskiy-PRF [16] for $(\mathsf{PRFKGen}_{\mathbb{G}}, \mathsf{PRF}_{\mathbb{G}})$, and with the proof-protocols and "lazy" $\mathsf{PRP}_{\mathbb{G}}$ described in Section 5, then by the security of the underlying building blocks we have the following corollary:

COROLLARY 4.1. *The (un)linkable pseudonym system described in Section 4 and instantiated as described above, securely realizes* $\mathcal{F}_{\mathsf{nym}}$ *in the* $(\mathcal{F}_{CA}, \mathcal{F}_{CRS}, \mathcal{F}_{SMT})$*-hybrid model under the Symmetric eXternal Decision Diffie-Hellman (SXDH) assumption [3], the q-Decisional Diffie-Hellman Inversion assumption [7], and the unforgeability of the AGOT scheme (which holds in the generic group model).*

Efficiency.

With the primitives instantiated as stated above, we obtain the following efficiency figures, where $\mathsf{exp}_{\mathbb{G}}$ denotes an exponentiation in group $\mathbb{G}$ and pair stands for a pairing computation. Many of these exponentiations can be merged into multi-base exponentiations which allows to substantially optimize the computational complexity.

Converter $\mathcal{X}$

PseudonymGeneration :	$10(+1)\mathsf{exp}_{\mathbb{G}} + 1\,\mathsf{exp}_{\tilde{\mathbb{G}}}$
ConversionRequest :	$7\,\mathsf{exp}_{\mathbb{G}} + 4\,\mathsf{exp}_{\mathbb{G}_t} + 8\,\mathsf{pair}$
ConversionResponse :	$34\,\mathsf{exp}_{\mathbb{G}} + 2\,\mathsf{exp}_{\tilde{\mathbb{G}}}$

Server ($\mathcal{S}_A$ or $\mathcal{S}_B$)

PseudonymGeneration :	$4(+1)\mathsf{exp}_{\mathbb{G}} + 4\,\mathsf{pair}$
ConversionRequest :	$8\,\mathsf{exp}_{\mathbb{G}} + 4\,\mathsf{exp}_{\mathbb{G}_t} + 8\,\mathsf{pair}$
ConversionResponse :	$30\,\mathsf{exp}_{\mathbb{G}} + 5\,\mathsf{exp}_{\mathbb{G}_t} + 12\,\mathsf{pair}$

4.3 Honest-but-Curious Converter

Our protocol achieves very strong security against active attacks, tolerating even a fully corrupt converter. One might argue though that a converter in our system is at most of the honest-but-curious type, i.e., the converter will always perform the protocol correctly but might aim at exploiting the information it sees or loose its data. Then, it will be sufficient to consider a weaker model where the converter will either be non-corrupted or of such honest-but-curious type. Regarding servers, considering active attacks is less debatable, however. Indeed, our pseudonym system can be used by a multitude of servers, possibly from private and public domains, and thus security should hold against servers that behave entirely malicious (as in our notion).

If one is willing to assume the weaker honest-but-curious adversary model for the converter, one can easily derive a more light-weight version from our protocol. Roughly, all parts where the converter proves correctness of its computations can be omitted. We now briefly sketch the necessary changes to our protocol and their impact on the efficiency numbers.

Pseudonym Generation.

In the pseudonym generation, the proof generation π_{nym} by the converter and the verification of π_{nym} and received signature σ_{nym} by the server $\mathcal{S}_A$ can be omitted. This reduces the complexity of the converter's part to $6\mathsf{exp}_{\mathbb{G}} + 1\,\mathsf{exp}_{\tilde{\mathbb{G}}}$ and $\mathcal{S}_A$ has to perform no exponentiation or pairing anymore.

$\underline{\mathsf{ConversionRequest:}}$ The server $\mathcal{S}_A$ requests a conversion of pseudonym $nym_{i,A}$ towards server $\mathcal{S}_B$.

Step1. Upon input $(\mathsf{CONVERT}, sid, qid, nym_{i,A}, \mathcal{S}_B)$**, Server** $\mathcal{S}_A$ **computes and sends request:**

a) Retrieve $(sid, nym_{i,A}, \sigma_{nym})$ for $nym_{i,A}$ and abort if no such record exists.
b) Compute $xnym_{i,A} \leftarrow \mathsf{PRP}^{-1}_{\mathbb{G}}(k_A, nym_{i,A})$, $C \xleftarrow{\$} \mathsf{Enc}_{\mathbb{G}}(epk_{\mathcal{X}}, \mathsf{Enc}_{\mathbb{G}}(epk_B, xnym_{i,A}))$, and $\sigma_C \leftarrow \mathsf{Sign}(ssk_A, (sid, qid, C))$.
c) Prove knowledge of a converter's signature σ_{nym} on the underlying $xnym_{i,A}$ and under key $spk_{\mathcal{X},\mathcal{A}}$:

$$\pi_A \xleftarrow{\$} \text{NIZK}\{(\underline{xnym_{i,A}}, \underline{\sigma_{nym}}) : \mathsf{Vf}_{\mathbb{G}}(spk_{\mathcal{X},\mathcal{A}}, \sigma_{nym}, xnym_{i,A}) = 1 \ \wedge \ C = \mathsf{Enc}_{\mathbb{G}}(epk_{\mathcal{X}}, \mathsf{Enc}_{\mathbb{G}}(epk_B, xnym_{i,A}))\}(sid, qid).$$

d) Send $(sid, qid, C, \pi_A, \sigma_C, \mathcal{S}_B)$ via $\mathcal{F}_{\text{SMT}}$ to $\mathcal{X}$ and end with no output.

Step2. Upon receiving $(sid, qid, C, \pi_A, \sigma_C, \mathcal{S}_B)$ **from** $\mathcal{S}_A$**,** $\mathcal{X}$ **verifies request and asks for permission to proceed:**

a) Verify that $\mathsf{Vf}(spk_A, \sigma_C, (sid, qid, C)) = 1$ and π_A is correct w.r.t. $spk_{\mathcal{X},\mathcal{A}}$ and the received ciphertext C.
b) Store $(\mathsf{convert}, sid, qid, C, \pi_A, \sigma_C, \mathcal{S}_A, \mathcal{S}_B)$ and output $(\mathsf{CONVERT}, sid, qid, \mathcal{S}_A, \mathcal{S}_B)$

$\underline{\mathsf{ConversionResponse:}}$ The converter $\mathcal{X}$ and server $\mathcal{S}_B$ blindly convert the encrypted pseudonym into $nym_{i,B}$.

Step1. Upon input $(\mathsf{PROCEED}, sid, qid)$**,** $\mathcal{X}$ **blindly derives the encrypted pseudonym** $xnym_{i,B}$ **for** $\mathcal{S}_B$**:**

a) Retrieve the conversion record $(\mathsf{convert}, sid, qid, C, \pi_A, \sigma_C, \mathcal{S}_A, \mathcal{S}_B)$ for qid and abort if no such record exists.
b) Compute $C' \leftarrow \mathsf{Dec}_{\mathbb{G}}(esk_{\mathcal{X}}, C)$ and $C'' \xleftarrow{\$} (C' \odot \mathsf{Enc}_{\mathbb{G}}(epk_B, 1))^{\Delta}$ where $\Delta \leftarrow x_B/x_A \pmod q$.
c) Sign the encrypted pseudonym using the secret key $ssk_{\mathcal{X},B}$ for $\mathcal{S}_B$ as $\overline{\sigma}_{nym} \xleftarrow{\$} \mathsf{EncSign}_{\mathbb{G}}(ssk_{\mathcal{X},B}, epk_B, C'')$.
d) Prove correctness of the computation of C'' and $\overline{\sigma}_{nym}$ in $\pi_{\mathcal{X}}$:

$$\pi_{\mathcal{X}} \xleftarrow{\$} \text{NIZK}\{(\Delta, C', ssk_{\mathcal{X},B}, esk_{\mathcal{X}}) : \overline{\sigma}_{nym} = \mathsf{EncSign}_{\mathbb{G}}(ssk_{\mathcal{X},B}, epk_B, C'') \ \wedge$$
$$C' = \mathsf{Dec}_{\mathbb{G}}(esk_{\mathcal{X}}, C) \ \wedge \ C'' = (C' \odot \mathsf{Enc}_{\mathbb{G}}(epk_B, 1))^{\Delta} \ \wedge \ y_A^{\Delta} = y_B\}(sid, qid).$$

e) Send $(sid, qid, C, C'', \sigma_C, \overline{\sigma}_{nym}, \pi_A, \pi_{\mathcal{X}}, S_A)$ via $\mathcal{F}_{\text{SMT}}$ to $\mathcal{S}_B$.

Step2. Upon receiving $(sid, qid, C, C'', \sigma_C, \overline{\sigma}_{nym}, \pi_A, \pi_{\mathcal{X}}, S_A)$ **from** $\mathcal{X}$**,** $\mathcal{S}_B$ **derives its local pseudonym** $nym_{i,B}$**:**

a) Verify that $\mathsf{Vf}(spk_A, \sigma_C, (sid, qid, C)) = 1$, π_A is correct w.r.t. $spk_{\mathcal{X},\mathcal{A}}, C$ and $\pi_{\mathcal{X}}$ is correct w.r.t. C''.
b) Compute $xnym_{i,B} \leftarrow \mathsf{Dec}_{\mathbb{G}}(esk_B, C'')$ and $\sigma_{nym} \leftarrow \mathsf{DecSign}_{\mathbb{G}}(esk_B, spk_{\mathcal{X},B}, \overline{\sigma}_{nym})$.
c) Verify that $\mathsf{Vf}_{\mathbb{G}}(spk_{\mathcal{X},B}, \sigma_{nym}, xnym_{i,B}) = 1$ and derive the final pseudonym as $nym_{i,B} \leftarrow \mathsf{PRP}_{\mathbb{G}}(k_B, xnym_{i,B})$.
d) Store $(sid, nym_{i,B}, \sigma_{nym})$ and end with output $(\mathsf{CONVERTED}, sid, qid, \mathcal{S}_A, nym_{i,B})$.

Figure 5: Conversion Request and Response Protocol

Conversion Request.

The changes to the conversion protocol are slightly more complex. When $\mathcal{S}_A$ prepares its request, we can remove the outer encryption layer of C and omit the signature σ_C. Both allowed $\mathcal{X}$ to forward $\mathcal{S}_A$'s request in a blind yet verifiable manner to $\mathcal{S}_B$. Relying on an honest-but-curious converter, this is not needed anymore. Overall, the complexity in the conversion request decreases to $5\,\mathsf{exp}_{\mathbb{G}} + 4\,\mathsf{exp}_{\mathbb{G}_t} + 8\,\mathsf{pair}$ for $\mathcal{S}_A$ and $3\,\mathsf{exp}_{\mathbb{G}} + 4\,\mathsf{exp}_{\mathbb{G}_t} + 8\,\mathsf{pair}$ for $\mathcal{X}$.

Conversion Response.

When the converter computes its conversion response, we can omit the proof $\pi_{\mathcal{X}}$. Further, $\mathcal{X}$ does not have to forward the proof π_A to $\mathcal{S}_B$ as this was needed in the security proof only when the converter was corrupt. Also the ciphertext C needs no longer to be forwarded to $\mathcal{S}_B$ and in fact *should* not be forwarded, as we just changed C to be a direct encryption of $nym_{i,A}$ under $\mathcal{S}_B'$s key. Overall, the only values sent from $\mathcal{X}$ to $\mathcal{S}_B$ are now $(sid, qid, C'', \overline{\sigma}_{nym}, S_A)$. Consequently, also the part of the receiving server $\mathcal{S}_B$ gets more light-weight: it does neither have to verify π_A, $\pi_{\mathcal{X}}$, or σ_C anymore. $\mathcal{S}_B$'s verification of the decrypted signature σ_{nym} on the converted pseudonym becomes obsolete, too. This significantly reduces the overall complexity of the response protocol to $23\,\mathsf{exp}_{\mathbb{G}} + 1\,\mathsf{exp}_{\tilde{\mathbb{G}}}$ for $\mathcal{X}$ and $3\,\mathsf{exp}_{\mathbb{G}}$ for $\mathcal{S}_B$.

5. CONCRETE INSTANTIATIONS

In this section we describe how to instantiate the different proofs used in our protocol, assuming that the ElGamal encryption scheme [17] is used for $(\mathsf{EncKGen}_{\mathbb{G}}, \mathsf{Enc}_{\mathbb{G}}, \mathsf{Dec}_{\mathbb{G}})$, AGOT+ signatures (as defined in Section 3.5) for $(\mathsf{SigKGen}_{\mathbb{G}}, \mathsf{Sign}_{\mathbb{G}}, \mathsf{EncSign}_{\mathbb{G}}, \mathsf{DecSign}_{\mathbb{G}}, \mathsf{Vf}_{\mathbb{G}})$ and the Dodis-Yampolskiy [16] construction for the verifiable pseudorandom function $(\mathsf{PRFKGen}_{\mathbb{G}}, \mathsf{PRF}_{\mathbb{G}})$. The concrete instantiations of the standard signature and the PRP have no influence on the proofs, as they don't appear in any of the proven statements. We also describe some optimisations for computing the nested encryption C and derivation of the ciphertext C'' which enhance the efficiency of our scheme.

Our instantiation requires that $\mathcal{F}_{\text{CRS}}$ provides all parties instead of the single group $\mathbb{G}$ with three groups $\mathbb{G} = \langle g \rangle$, $\tilde{\mathbb{G}} = \langle \tilde{g} \rangle$, $\mathbb{G}_t$ of prime order q, and a bilinear map $e : \mathbb{G} \times \tilde{\mathbb{G}} \to \mathbb{G}_t$. Those are generated as $(q, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_t, e, g, \tilde{g}) \xleftarrow{\$} \mathcal{G}(1^{\tau})$. For the system parameters of the AGOT(+) scheme also an additional random group element $x \xleftarrow{\$} \mathbb{G}$ is included in the CRS. Finally, to achieve online extractability for the NIZK proofs, we require that the CRS further contains a random public key $\hat{y} \in \mathbb{G}$. In the security proof, the simulator will choose a random $\hat{x} \in \mathbb{Z}_q$ and set $\hat{y} \leftarrow g^{\hat{x}}$, which allows to efficiently extract the necessary values as described in the preliminaries.

Overall, the CRS in our scheme then has the form $crs = (q, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_t, e, g, \tilde{g}, x, \hat{y})$. The converter's keys for the dual-mode signature (for each server $\mathcal{S}_A \in \mathbf{S}$) have the form $(spk_{\mathcal{X},A} = y_{\mathcal{X},A}, ssk_{\mathcal{X},A} = v_{\mathcal{X},A})$ and for the ElGamal encryption we have $(epk_{\mathcal{X}} = \bar{y}_{\mathcal{X}}, esk_{\mathcal{X}} = \bar{x}_{\mathcal{X}})$.

5.1 Pseudonym Generation

In π_{nym} a converter has to prove that it has generated its pseudonym contribution $xnym_{i,A}$ correctly. If the pseudonym is not anonymous ($anon = 0$), it also includes a proof that the core identifier $z_i = \mathsf{PRF}_{\mathbb{G}}(x_{\mathcal{X}}, uid_i)$ was computed correctly.

If the flag $anon = 1$, the proof π_{nym} is instantiated as follows: first compute an ElGamal encryption of z_i under the CRS key as $Z = (Z_1, Z_2) \leftarrow (\hat{y}^r, z_i g^r)$ with a randomly chosen $r \overset{\$}{\leftarrow} \mathbb{Z}_q$. Then compute the proof π'_{nym}:

$$\pi'_{nym} \overset{\$}{\leftarrow} \mathrm{SPK}\{(x'_A, r) :\ Z_1 = \hat{y}^r \ \wedge\ Z_2 = g^r xnym_{i,A}^{x'_A} \ \wedge \\ g = y_A^{x'_A}\ \}(sid, g, y_A, xnym_{i,A}, Z_1, Z_2)$$

and output $\pi_{nym} \leftarrow (Z, \pi'_{nym})$. For the analysis of this proof notice that $z_i = xnym_{i,A}^{1/x_A} = xnym_{i,A}^{x'_A}$.

If the flag $anon = 0$, then π_{nym} is instantiated as:

$$\pi_{nym} \overset{\$}{\leftarrow} \mathrm{SPK}\{(x'_A, x'_{\mathcal{X}}) :\ 1 = g^{x'_{\mathcal{X}}} y_{\mathcal{X}}^{-x'_A} \ \wedge\ g = y_A^{x'_A} \ \wedge \\ g = (xnym_{i,A}^{uid_i})^{x'_A} xnym_{i,A}^{x'_{\mathcal{X}}}\}(sid, g, y_A, xnym_{i,A}, y_{\mathcal{X}}, uid_i)$$

where $y_{\mathcal{X}}$ is part of the converter's public key. Let us analyse the latter proof. The first term established that $x'_{\mathcal{X}} = x'_A x_{\mathcal{X}}$, the second one that $x'_A = 1/x_A$ and the third term that $xnym_{i,A} = (g^{1/(uid_i + x_{\mathcal{X}})})^{x_A}$.

5.2 Conversion Request

In a conversion request, the server $\mathcal{S}_A$ has to prove that it knows a converter's signature on the inner pseudonym $xnym_{i,A}$, which it provided in double encrypted form $C = \mathsf{Enc}_{\mathbb{G}}(epk_{\mathcal{X}}, \mathsf{Enc}_{\mathbb{G}}(epk_B, xnym_{i,A}))$.

We start with the description of how the double encryption $C = \mathsf{Enc}_{\mathbb{G}}(epk_{\mathcal{X}}, \mathsf{Enc}_{\mathbb{G}}(epk_B, xnym_{i,A}))$ is instantiated. We already apply some optimizations and extend the ciphertext such that it allows for the online extraction of the pseudonym and its signature in the proof π_A.

Let $esk_{\mathcal{X}} = \bar{x}_{\mathcal{X}}$ and $epk_{\mathcal{X}} = \bar{y}_{\mathcal{X}}$ be the encryption key pairs of the converter and $esk_B = \bar{x}_B$ and $epk_B = \bar{y}_B$ for server $\mathcal{S}_B$. Let $\hat{y}$ be the public key in the CRS. Then C is computed as an extended ElGamal encryption as

$$C := (C_0, C_1, C_2, C_3) \leftarrow (\hat{y}^{r_1+r_2}, \bar{y}_B^{r_1}, \bar{y}_{\mathcal{X}}^{r_2}, g^{r_1+r_2} xnym_{i,A})$$

with $r_1, r_2 \overset{\$}{\leftarrow} \mathbb{Z}_q$. Let (r, s, r, w) be the converter's signature on $xnym_{i,A}$.

Then, the proof π_A if realized as follows: First the server $\mathcal{S}_A$ randomizes the signature $\sigma = (r, s, t, w)$ for key $spk_{\mathcal{X},A} = y_{\mathcal{X},A}$ by picking a random $u' \overset{\$}{\leftarrow} \mathbb{Z}_q^*$ and computes $\sigma' = (r', s', t', w')$ as

$$r' \leftarrow r^{u'}, \quad s' \leftarrow s^{1/u'}, \quad t' \leftarrow (tw^{(u'-1)})^{1/u'^2}.$$

Then it computes the proof

$$\pi'_A \overset{\$}{\leftarrow} \mathrm{SPK}\{(r_1, r_2, \nu_1, \nu_2) :\ C_0 = \hat{y}^{r_1+r_2} \ \wedge \\ C_1 = \bar{y}_B^{r_1} \ \wedge\ C_2 = \bar{y}_{\mathcal{X}}^{r_2} \ \wedge\ S_1 = \hat{y}^{\nu_1} \ \wedge\ T_1 = \hat{y}^{\nu_2} \ \wedge \\ e(x, \tilde{g})e(C_3, y_{\mathcal{X},A})/e(S_2, r') = e(g, y_{\mathcal{X},A})^{r_1+r_2} e(g, r')^{-\nu_1} \ \wedge \\ e(g, \tilde{g})e(S_2, y_{\mathcal{X},A})/e(T_2, r') = e(g, y_{\mathcal{X},A})^{\nu_1} e(g, r')^{-\nu_2} \\ \}(sid, qid, crs, C, r', S, T, w, y_{\mathcal{X},A}, \bar{y}_{\mathcal{X}}, \bar{y}_B),$$

where $S = (S_1, S_2) = (\hat{y}^{\nu_1}, g^{\nu_1} s')$ and $T = (T_1, T_2) = (\hat{y}^{\nu_2}, g^{\nu_2} t')$ are (ordinary) ElGamal encryptions under the CRS key that make this proof online extractable. It outputs $\pi_A = (\pi'_A, S, T, r')$. The analysis of this proof is given in the full version of the paper.

5.3 Conversion Response

Let us now detail how the converter computes the ciphertext $C'' = (C''_1, C''_2)$ and the proof $\pi_{\mathcal{X}}$ in which $\mathcal{X}$ proves that it derived and signed C'' (containing the translated pseudonym) correctly.

Given the ciphertext $C = (C_0, C_1, C_2, C_3)$, the converter computes $C'_2 \leftarrow C_3/C_2^{1/\bar{x}_{\mathcal{X}}}$, $C''_2 \leftarrow ({C'_2}^{x_B/x_A})g^r$, and $C''_1 \leftarrow C_1^{x_B/x_A}\bar{y}_B^r$, with $r \overset{\$}{\leftarrow} \mathbb{Z}_q$. Let $\Delta = x_B/x_A \pmod q$. Notice that (C_1, C'_2) is an encryption of $xnym_{i,A}$ under $\bar{y}_B$ (provided $\mathcal{S}_A$ computed C honestly) and that we have $C''_2 = (C_3/C_2^{1/\bar{x}_{\mathcal{X}}})^{\Delta} g^r$. Thus, $C'' = (C''_1, C''_2)$ is an encryption of $xnym_{i,B}$ under $\bar{y}_B$.

Now, the converter computes the signature on the ciphertext (C''_1, C''_2) and for signing key $ssk_{\mathcal{X},B} = v_{\mathcal{X},B}$ (with public key $spk_{\mathcal{X},B} = y_{\mathcal{X},B}$) . Choose a random $u, \rho_1, \rho_2 \overset{\$}{\leftarrow} \mathbb{Z}_q^*$, and compute the (partially) encrypted signature $\bar{\sigma} = (r, S, T, w)$:

$$\begin{aligned} r &\leftarrow \tilde{g}^u, & w &\leftarrow g^{1/u} \\ S_1 &\leftarrow {C''_1}^{v_{\mathcal{X},B}/u} \bar{y}_B^{\rho_1}, & S_2 &\leftarrow ({C''_2}^{v_{\mathcal{X},B}} x)^{1/u} g^{\rho_1}, \\ T_1 &\leftarrow S_1^{v_{\mathcal{X},B}/u} \bar{y}_B^{\rho_2}, & T_2 &\leftarrow (S_2^{v_{\mathcal{X},B}} g)^{1/u} g^{\rho_2}. \end{aligned}$$

Output $\bar{\sigma} = (r, (S_1, S_2), (T_1, T_2), w)$, where (S_1, S_2) and (T_1, T_2) are encryptions under $\mathcal{S}_B$'s public key $\bar{y}_B$.

Then, the proof $\pi_{\mathcal{X}}$ that $\mathcal{X}$ computed and signed (C''_1, C''_2) correctly and is as follows:

$$\pi_{\mathcal{X}} \overset{\$}{\leftarrow} SPK\{(u', v', \rho_1, \rho_2, \Delta, r, p) : \tilde{g} = r^{u'} \ \wedge\ w = g^{u'} \ \wedge \\ 1 = y_{\mathcal{X},B}^{-u'} \tilde{g}^{v'} \ \wedge S_1 = {C''_1}^{v'} \bar{y}_B^{\rho_1} \ \wedge S_2 = {C''_2}^{v'} x^{u'} g^{\rho_1} \ \wedge \\ T_1 = S_1^{v'} \bar{y}_B^{\rho_2} \ \wedge\ T_2 = S_2^{v'} g^{u'} g^{\rho_2} \ \wedge y_B = y_A^{\Delta} \ \wedge \\ C''_1 = C_1^{\Delta} \bar{y}_B^{r} \ \wedge\ C''_2 = C_3^{\Delta} C_2^{p} g^r \ \wedge\ 1 = g^{\Delta} \bar{y}_{\mathcal{X}}^{p} \\ \}(sid, qid, crs, r, S, T, w, C''_1, C''_2, C, y_{\mathcal{X},B}, \bar{y}_{\mathcal{X}}, \bar{y}_B) \ .$$

The last term establishes that $p = -\Delta/\bar{x}_{\mathcal{X}}$. The last four terms show that the ciphertext (C''_1, C''_2) was computed correctly from (C_0, C_1, C_2, C_3) whereas all other terms show that the "encrypted" signature was computed correctly.

5.4 Instantiating $\mathsf{PRP}_{\mathbb{G}}$ via Lazy Sampling

Our scheme makes use of a pseudorandom permutation $\mathsf{PRP}_{\mathbb{G}}$ to let each server derive its final pseudonym $nym_{i,A}$ as $nym_{i,A} \leftarrow \mathsf{PRP}_{\mathbb{G}}(k_A, xnym_{i,A})$ and also re-obtain $xnym_{i,A}$ via $\mathsf{PRP}_{\mathbb{G}}^{-1}$ in a conversion request. However, we mainly introduced the $\mathsf{PRP}_{\mathbb{G}}$ for notational convenience. In fact, it is sufficient to choose a random $nym_{i,A} \overset{\$}{\leftarrow} \mathbb{G}$ whenever a fresh $xnym_{i,A}$ is received, and to keep a list $\mathcal{L}_{\mathsf{nym}}$ of the mapping $(nym_{i,A}, xnym_{i,A})$. Then, whenever $xnym_{i,A}$ appears again, $\mathcal{S}_A$ simply retrieves $nym_{i,A}$ from $\mathcal{L}_{\mathsf{nym}}$ and vice-versa.

6. CONCLUSION AND EXTENSIONS

We have presented a protocol that allows to maintain and exchange data in a decentralized manner, based on pseudonyms which are per se unlinkable but can be transformed from one server to another with the help of a central converter. Our protocol overcomes the typical privacy bottleneck of such a system as it performs the pseudonym generation and conversion only in a blind way. It also provides strong guarantees in terms of consistency and controllability even if the converter is corrupt.

An interesting area for future work is to detail the different approaches on how to securely provision the pseudonyms. For instance, one possibility would be to combine our system with privacy-enhancing credentials that contain the unique identifier uid_i and are given to the users. That could allow a user to obtain a particular pseudonym contribution $xnym_{i,A}$ from the converter, and later prove towards $\mathcal{S}_A$ that she is indeed the correct "owner" of $xnym_{i,A}$.

Roughly, the idea would be to modify the pseudonym generation to output also a commitment com to uid_i, e.g., $com = g^{uid}h^r$ for the random opening information r. The proof π_{nym} (for $anon = 1$) that is generated by the converter to ensure correctness of the pseudonym would be modified accordingly to

$$\pi_{nym} \xleftarrow{\$} \text{NIZK}\{(x_{\mathcal{X}}, x_A, uid_i, r) : z_i = \mathsf{PRF}_{\mathbb{G}}(x_{\mathcal{X}}, uid_i) \ \wedge$$
$$xnym_{i,A} = z_i^{x_A} \wedge y_A = g^{x_A} \wedge com = g^{uid_i}h^r\}(sid, com).$$

Then, a user could register with a server $\mathcal{S}_A$ by providing $xnym_{i,A}$, the proof π_{nym} and then prove to the server that she owns a credential with the same uid_i that is contained in the commitment com without revealing uid_i.

Another interesting extension are audit capabilities. In a pseudonym system with a fully trusted converter, the converter could keep a log file of all server requests and allow the user (or a trusted auditor) to monitor which entities correlated or exchanged his data. With the blind conversions in our system, such a central audit is not immediately possible anymore. It is an interesting open problem how to add such audit capabilities without harming the privacy properties of our system.

In a similar vein, it would be desirable to combine our pseudonym system with policy enforcement tools in a privacy-preserving manner. That is, allowing the user to specify which data exchanges are permitted and enable the converter to blindly check whether a received conversion request violates any user constraints.

Acknowledgements

This work was supported by the European Commission through the Seventh Framework Programme, under grant agreements #321310 for the PERCY grant and #318424 for the project FutureID.

7. REFERENCES

[1] H. Aamot, C. D. Kohl, D. Richter, and P. Knaup-Gregori. Pseudonymization of patient identifiers for translational research. *BMC Medical Informatics and Decision Making 13:75*, 2013.

[2] M. Abe, J. Groth, M. Ohkubo, and M. Tibouchi. Unified, minimal and selectively randomizable structure-preserving signatures. *TCC 2014.*

[3] G. Ateniese, J. Camenisch, S. Hohenberger, and B. de Medeiros. Practical group signatures without random oracles. ePrint Archive, Report 2005/385.

[4] B. Barak, Y. Lindell, and T. Rabin. Protocol initialization for the framework of universal composability. ePrint Archive, Report 2004/006.

[5] M. Barbaro and T. Zeller. A face is exposed for aol searcher no. 4417749. New York Times, 2006.

[6] O. Blazy, G. Fuchsbauer, D. Pointcheval, and D. Vergnaud. Signatures on randomizable ciphertexts. *PKC 2011.*

[7] D. Boneh and X. Boyen. Efficient selective-ID secure identity based encryption without random oracles. *EUROCRYPT 2004.*

[8] B. Elger, J. Iavindrasana, L. Iacono, H. Muller, N. Roduit, P. Summers, and J. Wright. Strategies for health data exchange for secondary, cross-institutional clinical research. *Comput Methods Programs Biomed: 99(3)*, 2010.

[9] J. Camenisch, A. Kiayias, and M. Yung. On the portability of generalized schnorr proofs. *EUROCRYPT 2009.*

[10] J. Camenisch and M. Stadler. Efficient group signature schemes for large groups. *CRYPTO 1997.*

[11] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. ePrint Archive, Report 2000/067.

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

[13] Belgian Crossroads Bank for Social Security. `http://www.ksz.fgov.be/`.

[14] F. de Meyer, G. de Moor, and L. Reed-Fourquet. Privacy protection through pseudonymisation in ehealth. *Stud Health Technol Inform: 141*, 2008.

[15] Y.-A. de Montjoye, L. Radaelli, V. K. Singh, and A. Pentland. Unique in the shopping mall: On the reidentifiability of credit card metadata. *Science 30:347*, 2015.

[16] Y. Dodis and A. Yampolskiy. A verifiable random function with short proofs and keys. *PKC 2005.*

[17] T. ElGamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *CRYPTO 1984.*

[18] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. *CRYPTO 1986.*

[19] G. Fuchsbauer. Commuting signatures and verifiable encryption. *EUROCRYPT 2011.*

[20] D. Galindo and E. R. Verheul. Microdata sharing via pseudonymizatio. *Joint UNECE/Eurostat work session on statistical data confidentiality*, 2007.

[21] A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets. *IEEE Symposium on Security and Privacy*, 2008.

[22] T. Neubauer and J. Heurix. A methodology for the pseudonymization of medical data. *Int J Med Inform: 80(3)*, 2010.

[23] D. Pointcheval and J. Stern. Security proofs for signature schemes. *EUROCRYPT 1996.*

[24] K. Pommerening, M. Reng, P. Debold, and S. Semler. Pseudonymization in medical research - the generic data protection concept of the tmf. *GMS Medizinische Informatik 1:17*, 2005.

[25] C.-P. Schnorr. Efficient identification and signatures for smart cards. *CRYPTO 1990.*