Privacy for Distributed Databases via (Un)linkable Pseudonyms*

Jan Camenisch Anja Lehmann

IBM Research – Zurich
{jca,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.

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

*This is the full version of the paper *(Un)linkable Pseudonyms for Governmental Databases*, which appeared at ACM CCS 2015.

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) [15], 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 sensitive 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 [23]. 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 [14], 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 cannot 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 [17] and in the Netflix and AOL incidents [27, 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, 28, 16, 20, 30]. 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 security definition is formulated in the Universal Composability (UC) 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 is to build pseudonyms by adding several layers of randomness to the unique 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 are never revealed 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 [22], 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.

Again, all actions by the converter must be verifiable, i.e., a server $\mathcal{S}_B$ can be assured that it receives the correctly translated pseudonym $nym_{i,B}$, but without learning the pseudonym $nym_{i,A}$ it was derived from. Also, a server $\mathcal{S}_A$ can only trigger conversions for pseudonyms $nym_{i,A}$ that it either directly received from $\mathcal{X}$ or has obtained via conversion responses.

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. For the sake of clarity, we also provide a more algorithmic definition of the guaranteed consistency features in Appendix A.

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 [13], 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 and qid respectively.

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 $nym_{i,A} \xleftarrow{\$} \mathbf{N}$,
     - 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 $nym_{i,B} \xleftarrow{\$} \mathbf{N}$,
     - 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 $sid = (sid', \mathcal{X}, \mathbf{S}, \mathbf{U}, \mathbf{N})$

- 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 $\mathcal{X}$ 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 $\mathcal{X}$ is 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.

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 cannot 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 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 generates 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 zero-knowledge proofs of knowledge of discrete logarithms and statements about them, we will follow the notation introduced by Camenisch and Stadler [12] 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 [21].

Often we use a more abstract notation for proofs, e.g., by $NIZK\{(w) : statement(w)\}$ we denote any zero-knowledge proof protocol of knowledge of a witness w such that the $statement(w)$ 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 $NIZK\{(\underline{w_1}, w_2) : statement(w_1, w_2)\}$ 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 [18] 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 [8]. 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 $i \leftarrow \mathsf{PRP}^{-1}_{\mathbb{G}}(k,z)$. 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 $(epk, esk) \xleftarrow{\$} \mathsf{EncKGen}_{\mathbb{G}}(1^\tau)$, where τ is a security parameter, an encryption algorithm $C \xleftarrow{\$} \mathsf{Enc}_{\mathbb{G}}(epk, m)$, with $m \in \mathbb{G}$, and a decryption algorithm $m \leftarrow \mathsf{Dec}_{\mathbb{G}}(esk, C)$. Sometimes we will make the randomness used in the encryption process explicit, in which case we will write $C \leftarrow \mathsf{Enc}_{\mathbb{G}}(epk, m, r)$, where r encodes all the randomness, i.e., $\mathsf{Enc}_{\mathbb{G}}(\cdot,\cdot,\cdot)$ is a deterministic algorithm.

We require further that the encryption scheme has an appropriate *homomorphic property*, namely that there is an efficient operation $\odot$ on ciphertexts such that, if $C_1 \in \mathsf{Enc}_{\mathbb{G}}(epk, m_1)$ and $C_2 \in \mathsf{Enc}_{\mathbb{G}}(epk, m_2)$, then $C_1 \odot C_2 \in \mathsf{Enc}_{\mathbb{G}}(epk, m_1 \cdot m_2)$. We 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 $esk \leftarrow \bar{x}$ and $epk \leftarrow \bar{y}$.

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

$\mathsf{Dec}_{\mathbb{G}}(esk, C)$: On input the secret key $esk = \bar{x}$ 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 Signature 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 $(spk, ssk) \xleftarrow{\$} \mathsf{SigKGen}(1^\tau)$, a signing algorithm $\sigma \xleftarrow{\$} \mathsf{Sign}(ssk, m)$, 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 [22] 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. [7] 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 $(epk, esk) \xleftarrow{\$} \mathsf{EncKGen}_{\mathbb{G}}(1^\tau)$ 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.

$\mathsf{Vf}_{\mathbb{G}}(spk, \sigma, m)$: 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 $(spk, ssk) \xleftarrow{\$} \mathsf{SigKGen}_{\mathbb{G}}(1^\tau)$, all $(epk, esk) \xleftarrow{\$} \mathsf{EncKGen}_{\mathbb{G}}(1^\tau)$, all $m \in \mathbb{G}$, and all random choices in $\mathsf{Sign}_{\mathbb{G}}(\cdot,\cdot)$, 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 $\mathsf{Exp}^{\mathsf{DMSIG\text{-}forge}}_{\mathcal{A},\mathsf{DMSIG},\mathsf{Enc}_{\mathbb{G}}}(\mathbb{G}, \tau)$:
$(spk, ssk) \xleftarrow{\$} \mathsf{SigKGen}(1^\tau)$
$\mathbf{L} \leftarrow \emptyset$; $\mathbf{C} \leftarrow \emptyset$
$(m^*, \sigma^*) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathsf{Sign}}(ssk,\cdot),\mathcal{O}_{\mathsf{Enc}}(\cdot,\cdot),\mathcal{O}_{\mathsf{EncSign}}(ssk,\cdot,\cdot)}(spk)$
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 $\sigma \xleftarrow{\$} \mathsf{Sign}_{\mathbb{G}}(ssk, m_i)$
where $\mathcal{O}_{\mathsf{Enc}}$ on input (epk_i, m_i):
run $C_i \xleftarrow{\$} \mathsf{Enc}_{\mathbb{G}}(epk_i, m_i)$ 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 $\overline{\sigma} \xleftarrow{\$} \mathsf{EncSign}_{\mathbb{G}}(ssk, epk_i, C_i)$
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

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

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 Appendix B.1 how such a proof for the AGOT scheme 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 secret key $ssk = v$ and a proper encryption $M = \mathsf{Enc}_{\mathbb{G}}(epk, m)$ of a message $m \in \mathbb{G}$ under epk, choose a random $u \xleftarrow{\$} \mathbb{Z}_q^*$, and output the (partially) encrypted signature $\bar{\sigma} = (r, S, T, w)$:

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

$\mathsf{DecSign}_{\mathbb{G}}(esk, spk, \bar{\sigma})$: Parse $\bar{\sigma} = (r, S, T, w)$, 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 Appendix B.3.

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 Appendix B.2 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.)

[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.

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 $\mathcal{F}_{\mathrm{CA}}$, a secure message transmission functionality $\mathcal{F}_{\mathrm{SMT}}$ (enabling authenticated and encrypted communication), and a common reference string functionality $\mathcal{F}_{\mathrm{CRS}}$ are available to all parties. For details of those functionalities we refer to [13]. $\mathcal{F}_{\mathrm{CRS}}$ 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 $\mathcal{F}_{\mathrm{CA}}$ 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 $\mathcal{F}_{\mathrm{CA}}$

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 $\mathcal{F}_{\mathrm{CA}}$

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 [13, 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 Section 5.6 how this assumption can be relaxed .

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

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}:

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

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

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

Step2. Upon receiving $(sid, xnym_{i,A}, \sigma_{nym}, \pi_{nym}, uid_i)$ **from converter** $\mathcal{X}$**, server** 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 π_{nym} is correct w.r.t. y_A and $y_{\mathcal{X}}$, uid_i (if $uid_i \neq \perp$).

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

c) Output $(\mathsf{NYMGEN}, sid, nym_{i,A}, uid_i)$.

Figure 4: Pseudonym 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.

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 $xnym_{i,A} \leftarrow \mathsf{PRP}_{\mathbb{G}}^{-1}(k_A, nym_{i,A})$, $C \xleftarrow{\$} \mathsf{Enc}_{\mathbb{G}}(epk_{\mathcal{X}}, \mathsf{Enc}_{\mathbb{G}}(epk_B, xnym_{i,A}))$, 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 $spk_{\mathcal{X},\mathcal{A}}$:

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

d) Send $(sid, qid, C, \pi_A, \sigma_C, \mathcal{S}_B)$ via $\mathcal{F}_{\mathrm{SMT}}$ 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 $(\mathsf{CONVERT}, sid, qid, \mathcal{S}_A, \mathcal{S}_B)$

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** $xnym_{i,B}$**:**

a) Retrieve the conversion record $(\mathsf{convert}, sid, qid, C, \pi_A, \sigma_C, \mathcal{S}_A, \mathcal{S}_B)$ for qid, 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 $\pi_{\mathcal{X}}$:

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

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

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

We now show that our protocol described above securely realizes the ideal functionality $\mathcal{F}_{\mathsf{nym}}$ defined in Section 2. The proof is given in Appendix C.

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 [31, 29] 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 [18] 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 [8], 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.

Active Security	Converter $\mathcal{X}$	Server ($\mathcal{S}_A$ or $\mathcal{S}_B$)
PseudonymGeneration :	$10(+1)\mathsf{exp}_{\mathbb{G}} + 1\,\mathsf{exp}_{\tilde{\mathbb{G}}}$	$\mathcal{S}_A$: $4(+1)\mathsf{exp}_{\mathbb{G}} + 4\,\mathsf{pair}$
ConversionRequest :	$7\,\mathsf{exp}_{\mathbb{G}} + 4\,\mathsf{exp}_{\mathbb{G}_t} + 8\,\mathsf{pair}$	$\mathcal{S}_A$: $8\,\mathsf{exp}_{\mathbb{G}} + 4\,\mathsf{exp}_{\mathbb{G}_t} + 8\,\mathsf{pair}$
ConversionResponse :	$34\,\mathsf{exp}_{\mathbb{G}} + 2\,\mathsf{exp}_{\tilde{\mathbb{G}}}$	$\mathcal{S}_B$: $30\,\mathsf{exp}_{\mathbb{G}} + 5\,\mathsf{exp}_{\mathbb{G}_t} + 8\,\mathsf{pair}$

HbC-Version (Sec. 4.3)	Converter $\mathcal{X}$	Server ($\mathcal{S}_A$ or $\mathcal{S}_B$)
PseudonymGeneration :	$6\,\mathsf{exp}_{\mathbb{G}} + 1\,\mathsf{exp}_{\tilde{\mathbb{G}}}$	$\mathcal{S}_A$: —
ConversionRequest :	$3\,\mathsf{exp}_{\mathbb{G}} + 4\,\mathsf{exp}_{\mathbb{G}_t} + 8\,\mathsf{pair}$	$\mathcal{S}_A$: $5\,\mathsf{exp}_{\mathbb{G}} + 4\,\mathsf{exp}_{\mathbb{G}_t} + 8\,\mathsf{pair}$
ConversionResponse :	$23\,\mathsf{exp}_{\mathbb{G}} + 2\,\mathsf{exp}_{\tilde{\mathbb{G}}}$	$\mathcal{S}_B$: $3\,\mathsf{exp}_{\mathbb{G}}$

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.

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. 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 [19] 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 [18] 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.

5.1 System Parameters & CRS

As already pointed out in Section 4, the signature scheme and encryption scheme need to be compatible with the algebraic group $\mathbb{G}$, i.e., to sign and encrypt elements from $\mathbb{G}$. For the dual-mode signature we use the AGOT+ scheme which requires us to use bi-linear maps, though. Indeed, to the best of our knowledge there seems to be no signature scheme to sign group elements that allows for efficient proofs of knowledge of a signature on a group element which does not require bilinear maps.

Thus, we require 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 $(q, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_t, e, g, \tilde{g}) \xleftarrow{\$} \mathcal{G}(1^\tau)$. We require that the Decisional Diffie-Hellman assumption holds for all the three groups (even in the presence of the bilinear map). This is called the Symmetric eXternal Decision Diffie-Hellman (SXDH) assumption [3] and indeed the absence of efficiently computable maps between $\mathbb{G}$ and $\tilde{\mathbb{G}}$ seems to be a more natural setting than the one where such maps exist.

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 have the form $(spk = y, ssk = v)$, keys for the ElGamal encryption are denoted as $(epk = \bar{y}, esk = \bar{x})$ and the converter's key for the PRF is $(x_{\mathcal{X}}, y_{\mathcal{X}})$.

5.2 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

$$\pi_{nym} \xleftarrow{\$} \text{NIZK}\{(x_A, \underline{z_i}) \ : \ xnym_{i,A} = z_i^{x_A} \ \wedge \ y_A = g^{x_A} \ \wedge \ z_i \in \mathbb{G}\}(sid) \ .$$

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 $r \xleftarrow{\$} \mathbb{Z}_q$. Then compute the proof π'_{nym}:

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

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

If the flag $anon = 0$, then the proof

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

is instantiated as follows:

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

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 $xnym_{i,A} = (g^{1/(uid_i + x_{\mathcal{X}})})^{x_A} = \mathsf{PRF}_{\mathbb{G}}(x_{\mathcal{X}}, uid_i)^{x_A}$.

5.3 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 pair of the converter and $esk_B = \bar{x}_B$ and $epk_B = \bar{y}_B$ the key pair 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 $r_1, r_2 \xleftarrow{\$} \mathbb{Z}_q$. Let (r, s, t, w) be the converter's signature on $xnym_{i,A}$.

Then, the proof

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

is 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 $u' \xleftarrow{\$} \mathbb{Z}_q^*$ 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

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

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 follows from the proof π_M (which proves knowledge of a AGOT signature on an encrypted message) given in Section B.1.

5.4 Conversion Response

Let us now detail the response that is produced by the converter $\mathcal{X}$, and proves that it had correctly computed a signature on the encrypted pseudonym.

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 $r \xleftarrow{\$} \mathbb{Z}_q$. 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 $u, \rho_1, \rho_2 \xleftarrow{\$} \mathbb{Z}_q^*$, 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

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

that $\mathcal{X}$ computed and signed (C_1'', C_2'') correctly and is as follows:

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

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.

5.5 Simulation-Sound Zero-Knowledge Proofs

The most efficient way to make the proof protocol concurrent zero-knowledge and simulation-sound is by the Fiat-Shamir transformation [21]. In this case, we will have to resort to the random-oracle model [6] for the security proof. To make the resulting non-interactive proofs simulation-sound, it suffices to let the prover include context information as an argument to the random oracle in the Fiat-Shamir transformation, such as the system parameters, *sid*, *qid*, and the protocol step in which the statement is being proven, so that the proof is resistant to a man-in-the-middle attack. In particular, notice that all the statements we require the parties to prove to each other, are proofs of membership (i.e., that some computation was done correctly) and *not* proofs of knowledge. Therefore, it is not necessary that the prover can be re-wound to extract the witnesses.

We note, however, that there are alternative methods one could employ instead to make Σ-protocols non-interactive that do not rely on the random oracle model (e.g., [26, 24, 10]). Unfortunately, these methods come with some performance penalty.

5.6 On Realizations for the Trusted Setup

In our protocol we assumed that the setup phase where all keys are generated is trusted. This is needed for our security proof to go through, as therein the simulator has to know the secret keys used for the generation and conversion of the values $xnym_{i,A}$. Thus, in fact,

we only need to ensure trusted setup for the keys $(x_{\mathcal{X}}, y_{\mathcal{X}})$ and $\{(x_A, y_A)\}_{\forall \mathcal{S}_A \in \mathbf{S}}$ which are generated by the converter. Given that the converter is the crucial authority in our system, this seems to be a reasonable assumption (recall that the main goal was to protect against a too curious converter, and the converter is still allowed to be fully corrupt in the pseudonym generation and conversion, only the setup phase must be trusted).

However, if this setup assumption should be relaxed, one could extend the converter's setup procedure such that he has to prove correctness of those keys. This can be done as follows: For each conversion key pair (x_A, y_A) for server $\mathcal{S}_A \in \mathbf{S}$, the converter additionally computes

$$\pi_{y_A} \xleftarrow{\$} \text{NIZK}\{(\underline{x_A}) : y_A = g^{x_A}\}(sid, y_A).$$

For the key $(x_{\mathcal{X}}, y_{\mathcal{X}})$ of the verifiable PRF we make use of its verifiable nature and let $\mathcal{X}$ also compute $z \leftarrow \mathsf{PRF}_{\mathbb{G}}(x_{\mathcal{X}}, 0)$ and

$$\pi_{y_{\mathcal{X}}} \xleftarrow{\$} \text{NIZK}\{(\underline{x_{\mathcal{X}}}) : z = \mathsf{PRF}_{\mathbb{G}}(x_{\mathcal{X}}, 0)\}(sid, y_{\mathcal{X}}, z).$$

Those proofs are then included in the converter's public key $pk_{\mathcal{X}}$ and allow the simulator to extract the secret keys. As here the values to be extracted are exponents, we can not use the ElGamal encryption with a CRS trapdoor as for the other proofs, but would have to apply a verifiable encryption scheme instead, e.g., [11].

Another solution to enforce trusted setup is to use distributed key-generation techniques [25] for $(x_{\mathcal{X}}, y_{\mathcal{X}})$ and all $\{(x_A, y_A)\}_{\forall \mathcal{S}_A \in \mathbf{S}}$. That is, those keys are then jointly generated by all servers and the converter, but still with only the converter obtaining the keys. Then, as long as at least one honest server is involved, the simulator can set the keys accordingly.

5.7 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 $nym_{i,A} \xleftarrow{\$} \mathbb{G}$ 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. Symmetric encryption is an alternative as well, with proper mapping between $\mathbb{G}$ and the domain and range of the symmetric cipher.

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., $com = g^{uid_i} h^r$ 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

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

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.

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*, LNCS, 2014.

[3] G. Ateniese, J. Camenisch, S. Hohenberger, and B. de Medeiros. Practical group signatures without random oracles. Cryptology ePrint Archive, Report 2005/385, 2005. `http://eprint.iacr.org/2005/385`.

[4] B. Barak, Y. Lindell, and T. Rabin. Protocol initialization for the framework of universal composability. Cryptology ePrint Archive, Report 2004/006, 2004. `http://eprint.iacr.org/2004/006`.

[5] M. Barbaro and T. Zeller. A face is exposed for aol searcher no. 4417749. New York Times, 2006. `http://www.nytimes.com/2006/08/09/technology/09aol.html`.

[6] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby, editor, *ACM CCS 93*. ACM Press, Nov. 1993.

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

[8] D. Boneh and X. Boyen. Efficient selective-ID secure identity based encryption without random oracles. *EUROCRYPT 2004*, 3027 of *LNCS*, 2004.

[9] J. Camenisch, A. Kiayias, and M. Yung. On the portability of generalized schnorr proofs. *EUROCRYPT 2009*, 5479 of *LNCS*, 2009.

[10] J. Camenisch, S. Krenn, and V. Shoup. A framework for practical universally composable zero-knowledge protocols. In D. H. Lee and X. Wang, editors, *ASIACRYPT 2011*, 7073 of *LNCS*, Dec. 2011.

[11] J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. *CRYPTO 2003*, 2729 of *LNCS*, 2003.

[12] J. Camenisch and M. Stadler. Efficient group signature schemes for large groups (extended abstract). *CRYPTO'97*, 1294 of *LNCS*, 1997.

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

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

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

[16] F. de Meyer, G. de Moor, and L. Reed-Fourquet. Privacy protection through pseudonymisation in ehealth. Stud Health Technol Inform: 141, pp.111-118, 2008.

[17] 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: vol. 347 no. 6221 pp. 536-539, 2015.

[18] Y. Dodis and A. Yampolskiy. A verifiable random function with short proofs and keys. *PKC 2005*, 3386 of *LNCS*, 2005.

[19] T. ElGamal. On computing logarithms over finite fields. *CRYPTO'85*, 218 of *LNCS*, 1986.

[20] 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), pp. 230-251, 2010.

[21] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. *CRYPTO'86*, 263 of *LNCS*, 1987.

[22] G. Fuchsbauer. Commuting signatures and verifiable encryption. *EUROCRYPT 2011*, 6632 of *LNCS*, 2011.

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

[24] J. A. Garay, P. D. MacKenzie, and K. Yang. Strengthening zero-knowledge protocols using signatures. *EUROCRYPT 2003*, 2656 of *LNCS*, 2003.

[25] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. *EUROCRYPT'99*, 1592 of *LNCS*, 1999.

[26] P. D. MacKenzie and K. Yang. On simulation-sound trapdoor commitments. *EUROCRYPT 2004*, 3027 of *LNCS*, 2004.

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

[28] T. Neubauer and J. Heurix. A methodology for the pseudonymization of medical data. Int J Med Inform: 80(3), pp. 190-204, 2010.

[29] D. Pointcheval and J. Stern. Security proofs for signature schemes. *EUROCRYPT'96*, 1070 of *LNCS*, 1996.

[30] 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.

[31] C.-P. Schnorr. Efficient identification and signatures for smart cards. *CRYPTO'89*, 435 of *LNCS*, 1990.

A Consistency of (Un)linkable Identifiers

For the sake of simplicitly, we now also provide a more algorithmic definition of the consistency properties guaranteed by our ideal functionality $\mathcal{F}_{\mathsf{nym}}$ described in Section 2. Roughly, an (un)linkable identifier system must implement two functions NymGen and $\mathsf{Convert}$. For the generation of a pseudonym $nym_{i,A}$ for user uid_i and server $\mathcal{S}_A$, the server and converter jointly compute the function:

$$nym_{i,A} \leftarrow \mathsf{NymGen}(sk_{\mathcal{X}}, sk_A, uid_i, \mathcal{S}_A),$$

where $sk_{\mathcal{X}}$ denotes the secret key of the converter $\mathcal{X}$, and sk_A the secret key of the server $\mathcal{S}_A$. As described in Section 2, the converter does not learn the produced $nym_{i,A}$ whereas $\mathcal{S}_A$ does not learn uid_i (unless $anon = 0$).

For the conversion of a pseudonym $nym_{i,A}$ from server $\mathcal{S}_A$ to a pseudonym $nym_{i,B}$ for server $\mathcal{S}_B$, both servers and the converter $\mathcal{X}$ then jointly compute the function

$$nym_{i,B} \leftarrow \mathsf{Convert}(sk_{\mathcal{X}}, sk_A, sk_B, nym_{i,A}, \mathcal{S}_A, \mathcal{S}_B).$$

Here, the input $nym_{i,A}$ is only known to $\mathcal{S}_A$ and the output $nym_{i,B}$ is only given to $\mathcal{S}_B$.

The consistency properties guaranteed by our ideal functionality are now as follows. We denote with $\mathbf{U}$ the space of user identifiers, and the pseudonym space with $\mathbf{N}$.

NymGen **is injective:** For all $sk_{\mathcal{X}}, sk_A$, and all $uid_i, uid_j \in \mathbf{U}$ where $uid_i \neq uid_j$, $\mathsf{NymGen}(sk_{\mathcal{X}}, sk_A, uid_i, \mathcal{S}_A) \neq \mathsf{NymGen}(sk_{\mathcal{X}}, sk_A uid_j, \mathcal{S}_A)$.

$\mathsf{Convert}$ **is transitive:** For all $sk_{\mathcal{X}}, sk_A, sk_B, sk_C$ and $nym_A \in \mathbf{N}$, it holds that if $nym_B \leftarrow \mathsf{Convert}(sk_{\mathcal{X}}, sk_A, sk_B, nym_A, \mathcal{S}_A, \mathcal{S}_B)$ and $nym_C \leftarrow \mathsf{Convert}(sk_{\mathcal{X}}, sk_B, sk_C, nym_B, \mathcal{S}_B, \mathcal{S}_C)$ then $nym_C = \mathsf{Convert}(sk_{\mathcal{X}}, sk_A, sk_C, nym_A, \mathcal{S}_A, \mathcal{S}_C)$.

NymGen **and** $\mathsf{Convert}$ **are consistent:** For all $sk_{\mathcal{X}}, sk_A, sk_B$ and $uid \in \mathbf{U}$ it holds that i) if $nym_A \leftarrow \mathsf{NymGen}(sk_{\mathcal{X}}, sk_A, uid, \mathcal{S}_A)$ and $nym_B \leftarrow \mathsf{Convert}(sk_{\mathcal{X}}, sk_A, sk_B, nym_A, \mathcal{S}_A, \mathcal{S}_B)$ then $nym_B = \mathsf{NymGen}(sk_{\mathcal{X}}, sk_B, uid, \mathcal{S}_B)$. Likewise, ii) if $nym_A \leftarrow \mathsf{NymGen}(sk_{\mathcal{X}}, sk_A, uid, \mathcal{S}_A)$ and $nym_B \leftarrow \mathsf{NymGen}(sk_{\mathcal{X}}, sk_B, uid, \mathcal{S}_B)$ then $nym_B = \mathsf{Convert}(sk_{\mathcal{X}}, sk_A, sk_B, nym_A, \mathcal{S}_A, \mathcal{S}_B)$.

Instantiation of NymGen **and** $\mathsf{Convert}$**.** Our protocol described in Section 4 implements NymGen and $\mathsf{Convert}$ as follows. The pseudonym generation function, that is jointly computed by $\mathcal{X}$ and $\mathcal{S}_A$, can be summarized as:

$$\mathsf{NymGen}(sk_{\mathcal{X}}, sk_A, uid_i, \mathcal{S}_A) = \mathsf{PRP}_{\mathbb{G}}(k_A, \mathsf{PRF}_{\mathbb{G}}(x_{\mathcal{X}}, uid_i)^{x_A}).$$

And the conversion function that is jointly computed by $\mathcal{S}_A, \mathcal{S}_B$ and $\mathcal{X}$ is:

$$\mathsf{Convert}(sk_{\mathcal{X}}, sk_A, sk_B, nym_{i,A}, \mathcal{S}_A, \mathcal{S}_B) = \mathsf{PRP}_{\mathbb{G}}(k_B, (\mathsf{PRP}_{\mathbb{G}}^{-1}(k_A, nym_{i,A}))^{x_B/x_A}).$$

It is easy to see that NymGen and $\mathsf{Convert}$ satisfy the consistency requirements stated above. Again, consistency is enforced by our ideal functionality, and thus, the consistency property of our scheme already follows from Theorem 4.1.

B Proofs for the AGOT+ Scheme

Here, we present the different proofs we require from the AGOT(+) scheme. In Section B.1, we explain how a signature owner can prove knowledge of signature on an encrypted message. The signer's proof of a correct computation of a signature on an encrypted message is given in Section B.2. Finally, we prove in Section B.3 that the AGOT+ scheme is an unforgeable dual-mode signature scheme.

B.1 Proving Knowledge of a Signature on an Encrypted Message

Let $M = \mathsf{Enc}_{\mathbb{G}}(epk, m)$ be the encryption of a message $m \in \mathbb{G}$ and let (r, t, s, w) be a freshly randomized signature on m.

Now we want to prove knowledge of a signature on the encrypted message. Because the signature can be freshly randomized, we can reveal a part of it, namely r as it is an independent random value (the rest must not be revealed, it would leak information about the message), which makes our task easier.

On a high level, we will have to prove the following:

$$\pi_M \xleftarrow{\$} NIZK\{(m,s,t,\rho) : M = \mathsf{Enc}_{\mathbb{G}}(epk, m, \rho) \ \wedge \ e(x,\tilde{g}) = e(s,r)/e(m,y) \ \wedge \\ e(g,\tilde{g}) = e(t,r)/e(s,y)\}(e,x,g,\tilde{g},y,r,M,epk) \ ,$$

where the concrete realization depends on the encryption scheme used.

When we use the ElGamal scheme with a CRS trapdoor, the proof will look as follows, with $M = (M_0, M_1, M_2) = (\hat{y}^{\rho}, \bar{y}^{\rho}, g^{\rho} m)$ for a random $\rho \xleftarrow{\$} \mathbb{Z}_q$:

$$\pi_M \xleftarrow{\$} SPK\{(\rho,\nu_1,\nu_2,s,t) : M_1 = \bar{y}^{\rho} \ \wedge \ M_0 = \hat{y}^{\rho} \ \wedge \\ S_1 = \hat{y}^{\nu_1} \ \wedge \ T_1 = \hat{y}^{\nu_2} \ \wedge \ e(x,\tilde{g})e(M_2,y)/e(S_2,r) = e(g,y)^{\rho} e(g,r)^{-\nu_1} \ \wedge \\ e(g,\tilde{g})e(S_2,y)/e(T_2,r) = e(g,y)^{\nu_1} e(g,r)^{-\nu_2}\}(e,x,g,\tilde{g},y,r,M,S,T,\bar{y}),$$

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 that make this proof online extractable. Let us analyze this proof. From the first proof term, it follows that that $M_1^{1/\xi} = \bar{y}^{\rho/\xi} = g^{\rho}$, where $\xi = \log_g \bar{y}$, and similarly from the following terms that $M_0^{1/\varphi} = \hat{y}^{\rho/\varphi} = g^{\rho}$, $S_1^{1/\varphi} = \hat{y}^{\nu_1/\varphi} = g^{\nu_1}$, $T_1^{1/\varphi} = \hat{y}^{\nu_2/\varphi} = g^{\nu_2}$, where $\varphi = \log_g \hat{y}$. From the fifth proof term $e(x,\tilde{g})e(M_2,y)/e(S_2,r) = e(g,y)^{\rho} e(g,r)^{-\nu_1}$ we can derive that $e(x,\tilde{g})e(M_2 g^{-\rho}, y) = e(S_2 g^{-\nu_1}, r)$ and thus $e(x,\tilde{g})e(M_2/M_0^{1/\varphi}, y) = e(S_2/S_1^{1/\varphi}, r)$. From the last term the we can similarly derive that $e(g,\tilde{g})e(S_2/S_1^{1/\varphi}, y) = e(T_2/T_1^{1/\varphi}, r)$. Therefore, it holds that $(r, S_2/S_1^{1/\varphi}, T_2/T_1^{1/\varphi})$ is a valid AGOT signature on the message $M_2/M_0^{1/\varphi}$. That is, M encrypts a message on which the prover knows a signature and the signature and the message are extractable if the CRS is chosen so that φ is known to the extractor.

B.2 Proving Correctness of a Signature Computation

For our construction, we also require the signer to prove that she computed the signature on an encrypted message correctly:

$$\pi_\sigma \xleftarrow{\$} NIZK\{(u',v',\rho_1,\rho_2) : \tilde{g} = r^{u'} \ \wedge \ S = M^{v'} \odot \mathsf{Enc}_{\mathbb{G}}(epk, x, \rho_1)^{u'} \ \wedge \\ T = S^{v'} \odot \mathsf{Enc}_{\mathbb{G}}(epk, g, \rho_2)^{u'} \ \wedge \ w = g^{u'} \ \wedge \\ 1 = y^{-v'} \tilde{g}^{u'}\}(e,x,g,\tilde{g},y,M,r,S,T,w,epk) \ .$$

Let us analyse this proof. Recall that $v = ssk = \log_{\tilde{g}} y$, thus the last term establishes that $v' \equiv vu' \mod q$. Setting $u = 1/u'$, it follows that this proofs indeed shows that (r, S, T, w) was computed correctly, given the homomorphic properties of the encryption scheme.

When we use the ElGamal scheme for the encryption scheme, we can simplify the encrypted signature process as well as the proof. Note that we use standard ElGamal here, i.e., without a CRS trapdoor as the proof does not have to be online extractable.

Let $M = (M_1, M_2) = (\bar{y}^{\rho}, g^{\rho} m)$ be an encrypted message that we want to sign as follows. Parse $ssk = v$, choose a random $u, \rho_1, \rho_2 \xleftarrow{\$} \mathbb{Z}_q^*$, and compute the (partially) encrypted signature $\bar{\sigma} = (r, (S_1, S_2), (T_1, T_2), w)$:

$$
\begin{aligned}
r &\leftarrow \tilde{g}^u, & S_1 &\leftarrow M_1^{v/u} \bar{y}^{\rho_1}, & S_2 &\leftarrow (M_2^v x)^{1/u} g^{\rho_1}, \\
T_1 &\leftarrow S_1^{v/u} \bar{y}^{\rho_2}, & T_2 &\leftarrow (S_2^v g)^{1/u} g^{\rho_2}, & w &\leftarrow g^{1/u}.
\end{aligned}
$$

The signer then outputs $\bar{\sigma} = (r, (S_1, S_2), (T_1, T_2), w)$ together with the proof π_σ which looks as follows:

$$
\begin{aligned}
\pi_\sigma \xleftarrow{\$} SPK\{(u', v', \rho_1, \rho_2) : \tilde{g} = r^{u'} \ \wedge \ S_1 = M_1^{v'} \bar{y}^{\rho_1} \ \wedge \\
S_2 = M_2^{v'} x^{u'} g^{\rho_1} \ \wedge \ T_1 = S_1^{v'} \bar{y}^{\rho_2} \ \wedge \ T_2 = S_2^{v'} g^{u'} g^{\rho_2} \ \wedge \\
w = g^{u'} \ \wedge \ 1 = y^{-v'} \tilde{g}^{u'}\}(e, x, g, \tilde{g}, y, M, r, S, T, w, epk) \ .
\end{aligned}
$$

B.3 Proof of Theorem 3.2

Here we prove that our extension af the AGOT signature scheme yields an unforgeable dual-mode signature scheme. Let us call the AGOT scheme augmented with $\mathsf{EncSign}_{\mathbb{G}}, \mathsf{DecSign}_{\mathbb{G}}$, the AGOT+ scheme.

Proof. A reduction of the AGOT+ scheme to the AGOT scheme is rather straightforward. We are given access to an AGOT-sign oracle $\mathcal{O}'_{\mathsf{Sign}}$ and need to provide the 3 oracles of the dual-mode unforgeabilty game, which are a signing oracle $\mathcal{O}_{\mathsf{Sign}}$, an encryption oracle $\mathcal{O}_{\mathsf{Enc}}$ and an encrypted sign oracle $\mathcal{O}_{\mathsf{EncSign}}$. This can be done as follows. Standard signing queries m to the oracle $\mathcal{O}_{\mathsf{Sign}}$ are forwarded to $\mathcal{O}'_{\mathsf{Sign}}$ and replied with the obtained answer. Encryption queries (epk, m) to $\mathcal{O}_{\mathsf{Enc}}$ are answered by $M \xleftarrow{\$} \mathsf{Enc}_{\mathbb{G}}(epk, m)$, and encrypted signing queries M to $\mathcal{O}_{\mathsf{EncSign}}$ are handled as follows. First, according to the definition, M is only accepted if it was an output of $\mathcal{O}_{\mathsf{Enc}}$, thus it is guaranteed that M is a proper encryption and also we can look up the corresponding plaintext m. We submit m to $\mathcal{O}'_{\mathsf{Sign}}$ and get a signature (r, s, t, w) and then answer the query with $\bar{\sigma} = (r, \mathsf{Enc}_{\mathbb{G}}(epk, s), \mathsf{Enc}_{\mathbb{G}}(epk, t), w)$.

It remains to argue that the adversary cannot distinguish between the real game and our simulated oracles. This boils down to argue that the "encrypted" signature $\bar{\sigma} = (r, \mathsf{Enc}_{\mathbb{G}}(epk, s), \mathsf{Enc}_{\mathbb{G}}(epk, t), w)$ as output by our oracle $\mathcal{O}_{\mathsf{EncSign}}$ has the same distribution as if it was computed with $\mathsf{EncSign}_{\mathbb{G}}(ssk, epk, M)$. For r and w this is obviously the case. Let us consider S and T. In the encrypted signing algorithm $\mathsf{EncSign}_{\mathbb{G}}$, the value S is computed as $S \leftarrow (M^v \odot \mathsf{Enc}_{\mathbb{G}}(epk, x))^{1/u}$. Because of the homomorphic property of $\mathsf{Enc}_{\mathbb{G}}$ and because M is well formed, this corresponds to $S = \mathsf{Enc}_{\mathbb{G}}(epk, (m^v x)^{1/u})$, which is the encryption of a signature value s with the correct distribution. Together with the fact that $\mathsf{Enc}_{\mathbb{G}}(epk, x)$ is a fresh encryption that is not revealed, S is distributed as a fresh encryption of $(m^v x)^{1/u}$, just as we do in the reduction. Note that this holds independent of the distribution of M. A similar argument holds for T and therefore the output of the simulated oracle $\mathcal{O}_{\mathsf{EncSign}}$ on input M has the same distribution as the one of $\mathsf{EncSign}_{\mathbb{G}}(ssk, epk, M)$. ■

C Proof of Theorem 4.1

We now prove that our protocol described in Section 4 securely realizes the ideal functionality $\mathcal{F}_{\mathsf{nym}}$ defined in Section 2, as stated in Theorem 4.1.

Our proof consists of a sequence of games that a challenger runs with a real-world adversary. The challenger plays the role of all honest parties, obtaining their inputs from and passing their outputs to the environment. We gradually modify the protocol, such that in

our final game we can make the transition to let the challenger run internally the ideal functionality $\mathcal{F}_{\mathsf{nym}}$ and simulate all messages based merely on the information he can obtain from $\mathcal{F}_{\mathsf{nym}}$.

C.1 Sequence of Games

We now describe each game hop and argue why the view of the environment does not significantly change.

GAME 0: The challenger simply executes the real protocol for all honest parties, thereby giving the environment the same view as in the real world.

GAME 1: In a first game, we replace all messages that are sent between two honest parties via $\mathcal{F}_{\text{SMT}}$ by dummy messages. That is, instead of invoking $\mathcal{F}_{\text{SMT}}$ on the real input, we merely input 1^{ℓ}, where ℓ stands for the message length and can be determined from the security parameter and the protocol description. The real protocol values are kept internally by the challenger, and are used by all honest parties. Clearly, this step remains indistinguishable to the environment by the property of $\mathcal{F}_{\text{SMT}}$.

GAME 2: In this game we abort whenever we (as honest converter or honest server $\mathcal{S}_B$) receive in a conversion request a valid signature σ_C from an honest server $\mathcal{S}_A$ on some ciphertext C which that server had never produced. Indistinguishability of this game trivially follows from the unforgeability of the standard signature scheme $(\mathsf{SigKGen}, \mathsf{Sign}, \mathsf{Vf})$.

GAME 3A: We now change the way how an honest converter computes the conversion response C'' for a corrupt server $\mathcal{S}_B$. The goal is to make the response independent of the received ciphertext C. In this GAME 3a, we consider the case where the ciphertext C stems from a corrupt server S_A, whereas GAME 3b deals with settings where S_A is honest.

When an honest converter receives a ciphertext $C \leftarrow \mathsf{Enc}_{\mathbb{G}}(epk_{\mathcal{X}}, \mathsf{Enc}_{\mathbb{G}}(epk_B, xnym_{i,A}))$ from a corrupt server with corresponding proof π_A, from now on does not decrypt $C' \leftarrow \mathsf{Dec}_{\mathbb{G}}(esk_{\mathcal{X}}, C)$ anymore to derive C''. Instead, we extract the inner plaintext $xnym_{i,A}$ of C from π_A and compute the requested pseudonym as $xnym_{i,B} \leftarrow xnym_{i,A}^{x_B/x_A}$ where x_B and x_A are the secret conversion values maintained by the converter. The final ciphertext C'' is then generated directly based on $xnym_{i,B}$ as $C'' \leftarrow \mathsf{Enc}_{\mathbb{G}}(epk_B, xnym_{i,B})$ and the corresponding proof $\pi_{\mathcal{X}}$ is simulated. Conditioned on C being indeed a proper nested encryption of a $xnym_{i,A}$, and due the fact that the computation of C'' in the real protocol also includes a randomization of the ciphertext C', the ciphertexts C'' generated directly from $xnym_{i,B}$ have the same distribution as when derived according to the protocol.

Thus, the indistinguishability of this game hop is guaranteed by the simulation soundness and simulatability of the proof system.

GAME 3B: We now do the same modification to the generation of C'' for requests coming from honest servers. However, here we do not extract $xnym_{i,A}$ from π_A. Instead, we create an internal conversion record $(\mathsf{convert}, sid, qid, xnym_{i,A}, \mathcal{S}_A, \mathcal{S}_B)$ whenever the request for some pseudonym $nym_{i,A}$ is initiated by an honest server $\mathcal{S}_A$. The converter then simply retrieves $xnym_{i,A}$ from the record and computes $C'' \leftarrow \mathsf{Enc}_{\mathbb{G}}(epk_B, xnym_{i,B})$ and $xnym_{i,B} \leftarrow xnym_{i,A}^{x_B/x_A}$ as above. Again, as the derivation of C'' in the real protocol includes a randomization of the ciphertext $C' \leftarrow \mathsf{Dec}_{\mathbb{G}}(esk_{\mathcal{X}}, C)$, this different computation of C'' is indistinguishable to the environment.

GAME 4: When an honest converter generates its pseudonym contribution $xnym_{i,A}$ for identifier uid_i and server $\mathcal{S}_A$, it from now on also internally stores the tuple $(\mathsf{nym}, uid_i, \mathcal{S}_A, xnym_{i,A})$. We also create pseudonym records when a new pseudonym is generated in a conversion request. That is, when the converter receives a conversion request $(sid, qid, C, \pi_A,$

$\sigma_C, \mathcal{S}_B$) it computes $xnym_{i,B}$ as described in the previous game, but now also creates a new record $(\mathsf{nym}, uid_i, \mathcal{S}_B, xnym_{i,B})$, where uid_i is taken from the nym record for $xnym_{i,A}$. If no record $(\mathsf{nym}, uid_i, \mathcal{S}_A, xnym_{i,A})$ for the extracted $xnym_{i,A}$ exists, we abort. In other words, we abort if the adversary tries to initiate a conversion request for a value which is not a pseudonym the converter had generated before.

Indistinguishability of this game hop follows from the unforgeability of the converter's dual-mode signature scheme as specified in Definition 3.1. More precisely, if the environment can distinguish this game hop, we can derive an adversary $\mathcal{B}$ breaking the unforgeability of the dual-mode signature as follows: when $\mathcal{B}$ receives a public key spk of the dual-mode signature as input, it sets $spk_{\mathcal{X},B} \leftarrow spk$ for some randomly chosen corrupt server $\mathcal{S}_B \in \mathbf{S}$. The other setup values are generated according to the protocol. Then, any pseudonym request from $\mathcal{S}_B$ is answered by first computing $xnym_{i,B} \leftarrow \mathsf{PRF}_{\mathbb{G}}(x_{\mathcal{X}}, uid_i)^{x_B}$ and then obtaining $\sigma_{nym} \leftarrow \mathcal{O}_{\mathsf{Sign}}(xnym_{i,B})$ where $\mathcal{O}_{\mathsf{Sign}}$ is the plaintext-signing oracle from the unforgeability game. Similarly, for every pseudonym request $(sid, qid, C, \pi_A, \sigma_C, \mathcal{S}_B)$ towards $\mathcal{S}_B$, $\mathcal{B}$ computes $xnym_{i,B}$ based on the extracted $xnym_{i,A}$ and sends $(epk_B, xnym_{i,B})$ to the encryption oracle $\mathcal{O}_{\mathsf{Enc}}$ with epk_B being the encryption key published by $\mathcal{S}_B$. The ciphertext output by $\mathcal{O}_{\mathsf{Enc}}$ is then used as C'' in the conversion response. The encrypted signature is obtained as $\overline{\sigma}_{nym} \leftarrow \mathcal{O}_{\mathsf{EncSign}}(epk_B, C'')$. When $\mathcal{B}$ eventually receives a conversion request $(sid, qid, C, \pi_B, \sigma_C, \mathcal{S}_C)$ from $\mathcal{S}_B$ towards a server $\mathcal{S}_C$, it extracts $xnym_{i,B}, \sigma_{nym}$ from proof π_B. If no pseudonym record $(\mathsf{nym}, uid_i, \mathcal{S}_B, xnym_{i,B})$ for $xnym_{i,B}$ exists, $\mathcal{B}$ outputs $xnym_{i,B}, \sigma_{nym}$ as its forgery. It is easy to see that the simulation of the converter's signature is perfect, and thus indistinguishability of this game hop follows from the unforgeability of the converter's dual-mode signature scheme $(\mathsf{SigKGen}_{\mathbb{G}}, \mathsf{Sign}_{\mathbb{G}}, \mathsf{EncSign}_{\mathbb{G}}, \mathsf{DecSign}_{\mathbb{G}}, \mathsf{Vf}_{\mathbb{G}})$.

Game 5: From now on, an honest converter will not convert pseudonyms anymore but rather use its internal records or create pseudonyms from scratch. That is, whenever getting a conversion request for some $xnym_{i,A}$ (which we extract as before) the honest converter no longer computes $xnym_{i,B} \leftarrow xnym_{i,A}^{x_B/x_A}$. Instead it looks up the underlying uid_i from $(\mathsf{nym}, uid_i, \mathcal{S}_A, xnym_{i,A})$ and takes $xnym_{i,B}$ from the corresponding $(\mathsf{nym}, uid_i, \mathcal{S}_B, xnym_{i,B})$ record for uid_i and the target server $\mathcal{S}_B$. If no such pseudonym record exists, a new record is created with $xnym_{i,B}$ generated from scratch as $xnym_{i,B} \leftarrow \mathsf{PRF}_{\mathbb{G}}(x_{\mathcal{X}}, uid_i)^{x_B}$. It is easy to see that $xnym_{i,A}^{x_B/x_A} = (\mathsf{PRF}_{\mathbb{G}}(x_{\mathcal{X}}, uid_i)^{x_A})^{x_B/x_A} = \mathsf{PRF}_{\mathbb{G}}(x_{\mathcal{X}}, uid_i)^{x_B}$ and thus this game does not change the environment's view.

Game 6: In this game we modify the computation of the pseudonyms, when done by an honest converter, such that they become independent of the identifier uid_i. Normally, $\mathcal{X}$ computes its pseudonym contribution for uid_i and server $\mathcal{S}_A$ as $xnym_{i,A} \leftarrow \mathsf{PRF}_{\mathbb{G}}(x_{\mathcal{X}}, uid_i)^{x_A}$. From now on, we will (via a series of hybrids) generate this inner pseudonym as $xnym_{i,A} \leftarrow z_i^{x_A}$ where $z_i \leftarrow g^{x_i}$ for a randomly chosen $x_i \xleftarrow{\$} \mathbb{Z}_q$, and simulate the corresponding proof π_{nym}. Note that we do the replacement of $\mathsf{PRF}_{\mathbb{G}}(x_{\mathcal{X}}, uid_i)$ by a random z_i only the first time a pseudonym is requested for a particular uid_i and then internally store the tuple (uid_i, z_i, x_i). For every further pseudonym based on uid_i we then reuse the same z_i. The reason to choose $z_i \leftarrow g^{x_i}$ instead of $z_i \xleftarrow{\$} \mathbb{G}$ is that we will make use of the discrete logarithm x_i in the next game in order to simulate pseudonyms without knowing a server's secret conversion key x_A.

The indistinguishability of this game follows from the PRF-property of $\mathsf{PRF}_{\mathbb{G}}$ and the zero-knowledge property of the proof system.

Game 7: We now take the next step and set the pseudonym contributions that are generated by an honest converter to fully random values in $\mathbb{G}$ (again, via a hybrid argument, replacing the pseudonyms for each combination of uid_i and $\mathcal{S}_A$ one by one). Thus, whenever the converter is asked to generate a pseudonym for uid_i and server $\mathcal{S}_A$, it returns $xnym_{i,A} \xleftarrow{\$} \mathbb{G}$ together with a correct signature σ_{nym} and simulated proof π_{nym}. Note that the conversion process is done solely based on internal records since Game 5, and thus, this modification has no impact on the conversion.

If an adversary can distinguish this game hop, we can derive an adversary that solves the decisional Diffie-Hellman problem. Let (g^a, g^b, g^c) denote a DDH challenge where the task is to determine whether $c = ab$ or not. Then, if the hybrid where we replace the pseudonym $xnym_{i,A}$ for uid_i and server $\mathcal{S}_A$ can be distinguished by the environment, we set $y_A \leftarrow g^a$, $z_i \leftarrow g^b$ and $xnym_{i,A} = g^c$. Thus, if (g^a, g^b, g^c) is indeed a DDH tuple, the pseudonym is still computed as in the previous game, whereas for $c \neq ab$ we would have replaced it by a random value. Note that we still have sufficient information to correctly derive other pseudonyms for $\mathcal{S}_A$ and uid_i. For every pseudonym $xnym_{i,B}$ based on the same uid_i but a different server $\mathcal{S}_B \neq \mathcal{S}_A$, we compute $xnym_{i,B} = z_i^{x_B}$ as before. For pseudonyms $xnym_{j,A}$ that should be based on $\mathcal{S}'_A s$ secret conversion key key x_A, but for a different $uid_j \neq uid_i$, we set $xnym_{j,A} \leftarrow y_A^{x_j}$ using the x_j values stored for uid_j since the last game.

GAME 8: In this game we let an honest server not compute any real protocol values anymore when doing a request via an honest converter. Again, this does not affect the conversion process, as that is done solely based on internal records since GAME 5. When the conversion is done towards a corrupt server $\mathcal{S}_B$, we then mimic the values the converter is supposed to forward from $\mathcal{S}_A$. That is, we derive $C \leftarrow \mathsf{Enc}_{\mathbb{G}}(epk_{\mathcal{X}}, 1)$ as a dummy encryption and also fake the corresponding proof π_A using the zero-knowledge simulator. However, we do complement C with a correct signature $\sigma_C \leftarrow \mathsf{Sign}(ssk_A, (sid, qid, C))$ using $\mathcal{S}_A$'s signing key. The zero-knowledge property of the proof scheme and the semantic security of the encryption scheme $(\mathsf{EncKGen}_{\mathbb{G}}, \mathsf{Enc}_{\mathbb{G}}, \mathsf{Dec}_{\mathbb{G}})$ ensure that this game hop has no significant impact on the environment's view. Note that we do not require CCA security here, as an honest converter does not decrypt any ciphertexts C since GAME 3 anymore. Further, since the proof π_A is also bound to the qid and includes a statement on the public key of $\mathcal{S}_A$, the adversary can not reuse the dummy ciphertext and fake proof in a different context.

GAME 9: In the previous games we mainly changed the protocol for the setting where the converter was honest. The current and following games now deal with protocol modifications we do when the convert is corrupt. From now on we let an honest server $\mathcal{S}_A$ whenever having to output a new pseudonym $nym_{i,A} \leftarrow \mathsf{PRP}_{\mathbb{G}}(k_A, xnym_{i,A})$ output a random value $nym_{i,A} \xleftarrow{\$} \mathbb{G}$ instead. To ensure consistency, we keep a list $\mathcal{L}_{\mathsf{nym}}$ of pairs $(nym_{i,A}, xnym_{i,A})$, and answer consistently with the same random $nym_{i,A}$ whenever the server has to provide an output for the same $xnym_{i,A}$. Similarly, whenever a conversion for pseudonym $nym_{i,A}$ is triggered we do not compute $xnym_{i,A} \leftarrow \mathsf{PRP}_{\mathbb{G}}^{-1}(k_A, nym_{i,A})$ anymore but obtain $xnym_{i,A}$ by looking up the corresponding $nym_{i,A}$ from $\mathcal{L}_{\mathsf{nym}}$. Indistinguishability of this game hop follows from the pseudorandomness property of the $\mathsf{PRP}_{\mathbb{G}}$.

GAME 10: In this game, we now get rid of the decryption an honest server $\mathcal{S}_B$ has to make when receiving a conversion response from a corrupt converter. To this end, we first create internal records for parts of the converter's secret keys, namely $x_{\mathcal{X}}$ and $\{x_A\}_{\forall \mathcal{S}_A \in \mathbf{S}}$ that are generated in the trusted setup. Thus, here we need the setup assumption, such that the simulator provide the keys to the converter and be privy of their secret keys. (When realizing the trusted setup via additional proofs $\pi_{y_{\mathcal{X}}}, \{\pi_{y_A}\}_{\forall \mathcal{S}_A \in \mathbf{S}}$, as described in Section 5.6 we would now extract the secret keys from those proofs.)

When an honest server $\mathcal{S}_B$ then receives a conversion response $(sid, qid, C, C'', \sigma_C, \overline{\sigma}_{nym}, \pi_A, \pi_{\mathcal{X}}, S_A)$ from a corrupt converter for a request made by a server $\mathcal{S}_A$, it does not decrypt C'' anymore but derives $xnym_{i,B}$ by computing $xnym_{i,B} \leftarrow xnym_{i,A}^{x_B/x_A}$ using the extracted keys x_B, x_A. If the request came from a corrupt server $\mathcal{S}_A$, then $\mathcal{S}_B$ obtains $xnym_{i,A}$ from π_A. For requests coming from an honest server, we don't extract $xnym_{i,A}$ from π_A. Instead we create an internal record $(\mathsf{convert}, sid, qid, xnym_{i,A}, \mathcal{S}_A, \mathcal{S}_B)$ when $\mathcal{S}_A$ initiates the request and let $\mathcal{S}_B$ simply look up the value $xnym_{i,A}$ from there.

We also don't let $\mathcal{S}_B$ "decrypt" the signature $\overline{\sigma}_{nym}$ anymore. The server $\mathcal{S}_B$ only verifies that the signature was correctly computed (which is proven in $\pi_{\mathcal{X}}$), but does not compute σ_{nym} for the derived pseudonym $xnym_{i,B}$. For any subsequent conversion request initiated

by $\mathcal{S}_B$ that would require to prove knowledge of such a σ_{nym} we merely simulate the corresponding part of the proof π_B.

Overall, indistinguishability of this game hop follows from the zero-knowledge property and simulation-soundness of the proof system.

GAME 11: Here, we change the way a ciphertext C is formed whenever an honest server $\mathcal{S}_A$ is requesting a conversion towards an honest server $\mathcal{S}_B$ via a corrupt converter $\mathcal{X}$. Namely, we replace the proper inner encryption $\mathsf{Enc}_{\mathbb{G}}(epk_B, xnym_{i,A})$ contained in C by a dummy encryption, i.e., we set $C \leftarrow \mathsf{Enc}_{\mathbb{G}}(epk_{\mathcal{X}}, \mathsf{Enc}_{\mathbb{G}}(epk_B, 1))$ and fake the corresponding proof π_A using the zero-knowledge simulator. This is a legitimate modification by the semantic security of the encryption scheme $(\mathsf{EncKGen}_{\mathbb{G}}, \mathsf{Enc}_{\mathbb{G}}, \mathsf{Dec}_{\mathbb{G}})$ (using a hybrid argument) and the zero-knowledge property of the proof system. Note that in the reduction to the semantic security of the encryption scheme we do not have a decryption oracle. However, in the last two games, we already got rid of all the decryption an honest $\mathcal{S}_B$ has to do.

GAME 12: In our final game we make the transition from letting the challenger run the "real" protocol (w.r.t. GAME 11) to letting him interact with the ideal functionality $\mathcal{F}$ and simulate all messages based solely on the information he can obtain from $\mathcal{F}$. The description of our simulator follows in Appendix C.2.

C.2 Simulator

We now complete our proof of Theorem 4.1 and show that we can construct a simulator sim such that for any environment $\mathcal{E}$ and adversary $\mathcal{A}$ that controls a certain subset of the parties, the view of the environment in the real world, when running the protocol (according to GAME 11 from Section C.1) with the adversary, is indistinguishable from its view in the ideal world where it interacts with the ideal functionality and the simulator (which corresponds to the final game from Section C.1).

We split the description of the simulator into two main cases, depending on whether the converter is corrupt or honest. In both cases, the behaviour in the subprotocols can further branch depending on the respective setting of corrupt and honest parties. The simulator will always play the role of all honest parties in the real protocol and store all records created by an honest party, according to the real protocol. We denote by "$\mathcal{S}_A$" the simulation of an honest server $\mathcal{S}_A$, and by "$\mathcal{X}$" a simulated honest converter.

C.2.1 Honest Converter

Pseudonym Generation. When sim receives $(\mathsf{NYMGEN}, sid, \mathcal{S}_A)$ from $\mathcal{F}$, it branches the simulation depending on $\mathcal{S}_A$ being honest or corrupt:

for honest server $\mathcal{S}_A$: Here "$\mathcal{X}$" merely simulates the communication to "$\mathcal{S}_A$", as it does not know the pseudonym or identifier "$\mathcal{S}_A$" is supposed to get. That is, "$\mathcal{X}$" invokes $\mathcal{F}_{\text{SMT}}$ only on a dummy message of the right length. When received by "$\mathcal{S}_A$", the simulator sends $(\mathsf{NYMGEN}, sid, \mathcal{S}_A, \bot)$ to $\mathcal{F}$ which triggers the delivery of the pseudonym in the ideal world.

for corrupt server $\mathcal{S}_A$: When $\mathcal{S}_A$ is corrupt, the simulator immediately sends $(\mathsf{NYMGEN}, sid, \mathcal{S}_A, \bot)$ to $\mathcal{F}$, receiving $(\mathsf{NYMGEN}, sid, nym_{i,A}, uid_i)$ from the ideal functionality. The honest converter "$\mathcal{X}$" then signs the ideal pseudonym as $\sigma_{nym} \leftarrow \mathsf{Sign}(ssk_{\mathcal{X},A}, nym_{i,A})$ and simulates the proof π_{nym} for $anon = 1$ if $uid = \bot$ and for $anon = 0$ otherwise. Thus, whenever we have to simulate a protocol step, we use the pseudonyms $nym_{i,A}$ we obtain from $\mathcal{F}$ as $xnym_{i,A}$ values in the protocol. The simulator then internally stores $(sid, nym_{i,A}, \mathcal{S}_A, \sigma_{nym})$ and sends $(sid, nym_{i,A}, \sigma_{nym}, \pi_{nym}, uid_i)$ via $\mathcal{F}_{\text{SMT}}$ to $\mathcal{S}_A$. Note that from now on, all pseudonyms $nym_{i,A}$ that are sent to the adversary are generated by the ideal functionality and thus random values in $\mathbb{G}$.

Pseudonym Conversion. The simulated honest converter now has to ensure that pseudonyms are translated consistently in the real world, even though it has distributed the random pseudonyms generated by the ideal functionality.

from honest to honest server: When the simulator receives $(\mathsf{CONVERT}, sid, qid, \mathcal{S}_A, \mathcal{S}_B)$ from $\mathcal{F}$, it simulates the conversion request between "$\mathcal{S}_A$" and "$\mathcal{X}$" by sending a dummy message via $\mathcal{F}_{\mathrm{SMT}}$ to "$\mathcal{X}$". When "$\mathcal{X}$" receives that message, sim also triggers the delivery of the message $(\mathsf{CONVERT}, sid, qid, \mathcal{S}_A, \mathcal{S}_B)$ to $\mathcal{X}$ in the ideal world. When the environment gave the ok to proceed, sim gets notified with the message $(\mathsf{PROCEED}, sid, qid)$ and then simulates the conversion completion towards "$\mathcal{S}_B$". Again, only a dummy message is sent via $\mathcal{F}_{\mathrm{SMT}}$ and when it arrives, sim sends $(\mathsf{PROCEED}, sid, qid, \bot)$ to $\mathcal{F}$ which triggers the output to $\mathcal{S}_B$ in the ideal world as well.

from honest to corrupt server: As above, the simulation starts when sim receives $(\mathsf{CONVERT}, sid, qid, \mathcal{S}_A, \mathcal{S}_B)$ from $\mathcal{F}$. "$\mathcal{S}_A$" then simulates the conversion request between "$\mathcal{S}_A$" and "$\mathcal{X}$" as in the case above. As here the target server $\mathcal{S}_B$ is corrupt, sim immediately replies with $(\mathsf{PROCEED}, sid, qid, \bot)$ towards $\mathcal{F}$ when it receives $(\mathsf{PROCEED}, sid, qid)$ from the functionality. The simulator then obtains the pseudonym $(\mathsf{CONVERTED}, sid, qid, \mathcal{S}_A, nym_{i,B})$ from $\mathcal{F}$, which allows him to prepare a correct conversion response towards $\mathcal{S}_B$. To this end, "$\mathcal{X}$" computes $C'' \xleftarrow{\$} \mathsf{Enc}_{\mathbb{G}}(epk_B, nym_{i,B})$, $\overline{\sigma}_{nym} \xleftarrow{\$} \mathsf{EncSign}_{\mathbb{G}}(ssk_{\mathcal{X},B}, epk_B, C'')$ and simulates the proof $\pi_{\mathcal{X}}$. The values "$\mathcal{X}$" is supposed to forward from "$\mathcal{S}_A$" are set to $C \leftarrow \mathsf{Enc}_{\mathbb{G}}(epk_{\mathcal{X}}, 1)$, $\sigma_C \leftarrow \mathsf{Sign}(ssk_A, (sid, qid, C))$ and π_A is simulated. Finally, "$\mathcal{X}$" sends $(sid, qid, C, C'', \sigma_C, \overline{\sigma}_{nym}, \pi_A, \pi_{\mathcal{X}}, S_A)$ via $\mathcal{F}_{\mathrm{SMT}}$ to $\mathcal{S}_B$ and ends.

from corrupt to honest server: In this case, the simulation starts when the converter "$\mathcal{X}$" in the real world receives $(sid, qid, C, \pi_A, \sigma_C, \mathcal{S}_B)$ from a corrupt server $\mathcal{S}_A$. If the tuple is valid, "$\mathcal{X}$" extracts $xnym_{i,A}$ from π_A and sends $(\mathsf{CONVERT}, sid, qid, xnym_{i,A}, \mathcal{S}_B)$ to $\mathcal{F}$ (recall that we used the pseudonyms $nym_{i,A}$ provided by $\mathcal{F}$ as $xnym_{i,A}$ values in the simulated protocol). When sim then receives $(\mathsf{PROCEED}, sid, qid)$ from $\mathcal{F}$, it simulates the response to "$\mathcal{S}_B$" by sending a dummy message via $\mathcal{F}_{\mathrm{SMT}}$ to "$\mathcal{S}_B$". When "$\mathcal{S}_B$" receives the message, the simulator also triggers the output in the ideal world by sending $(\mathsf{PROCEED}, sid, qid, \bot)$ to $\mathcal{F}$.

from corrupt to corrupt server: The simulation here is similar to the case above and starts when "$\mathcal{X}$" receives a message $(sid, qid, C, \pi_A, \sigma_C, \mathcal{S}_B)$ from a corrupt server $\mathcal{S}_A$. However, here "$\mathcal{X}$" has to provide a correct looking and consistent conversion response to the corrupt $\mathcal{S}_B$. That is, when sim receives $(\mathsf{PROCEED}, sid, qid)$ from $\mathcal{F}$ it immediately responds with $(\mathsf{PROCEED}, sid, qid, \bot)$ which will trigger the output $(\mathsf{CONVERTED}, sid, qid, \mathcal{S}_A, nym_{i,B})$ to sim. Having learned the target pseudonym $nym_{i,B}$, the simulator now computes a correct response in the real world as $C'' \xleftarrow{\$} \mathsf{Enc}_{\mathbb{G}}(epk_B, nym_{i,B})$, $\overline{\sigma}_{nym} \xleftarrow{\$} \mathsf{EncSign}_{\mathbb{G}}(ssk_{\mathcal{X},B}, epk_B, C'')$ and simulates $\pi_{\mathcal{X}}$. The honest converter then sends those values together with C, π_A, σ_C received from $\mathcal{S}_A$ to $\mathcal{S}_B$.

C.3 Corrupt Converter

Here most parts of the protocol are done by the corrupt converter, and the main task of the simulator is to reflect the learned information in the ideal functionality. Recall, that since Game 10, the simulator internally stores the secret keys $x_{\mathcal{X}}$ and $\{x_A\}_{\forall \mathcal{S}_A \in \mathbf{S}}$ of the converter, which stem from the trusted setup (or by extracting them for $\pi_{y_{\mathcal{X}}}, \{\pi_{y_A}\}_{\forall \mathcal{S}_A \in \mathbf{S}}$ which would be contained in $\mathcal{X}$'s public key if the trusted setup is realized with proofs for the correct keys as described in Section 5.6).

Pseudonym Generation (for honest server $\mathcal{S}_A$). As the generation of pseudonyms is initiated by the converter, our simulation starts when an honest server "$\mathcal{S}_A$" in the real world receives $(sid, nym_{i,A}, \sigma_{nym}, \pi_{nym}, uid_i)$ from the corrupt converter $\mathcal{X}$. When "$\mathcal{S}_A$" outputs $(\mathsf{NYMGEN}, sid, nym_{i,A}, uid_i)$, the simulator has to ensure that a corresponding pseudonym

record is created in the ideal functionality as well. If the generation was done verifiably, i.e., the received tuple contains $uid_i \neq \perp$, sim first checks if the uid_i belongs to a "dummy" identifier it has chosen before. That is, sim checks if there is a value $z_i \in \mathcal{L}_{\mathsf{dummy}}$ such that $z_i = \mathsf{PRF}_{\mathbb{G}}(x_{\mathcal{X}}, uid_i)$. If that is the case, sim sends $(\mathsf{ASSIGN}, sid, (\text{“rid“}||z_i), uid_i)$ to $\mathcal{F}$, ensuring that all pseudonyms that were previously linked to the dummy identifier $(\text{“rid“}||z_i)$ are now linked to the correct value uid_i. When this is done, sim also creates a pseudonym for $\mathcal{S}_A$ in $\mathcal{F}$ by sending $(\mathsf{NYMGEN}, sid, uid_i, \mathcal{S}_A, anon)$ to $\mathcal{F}$ where $anon = 0$ if the message in the real world contained the underlying uid_i.

In addition, we let sim also create corresponding pseudonym records in $\mathcal{F}$ for *all* corrupt servers $\mathcal{S}_B \in \mathbf{S}$. For those records, the simulator can provide the pseudonym value $nym^*_{i,B}$ since both $\mathcal{X}$ and $\mathcal{S}_B$ are corrupt. We derive those values using the knowledge of the real-world converter's secret key, which will allow us to ensure consistent simulation of conversion requests later on.

Depending on whether the pseudonym $nym_{i,A}$ was provided anonymously or not, we will either be able to link all pseudonyms to the correct uid_i, or to a dummy – yet consistent – identifier $(\text{“rid“}||z_i)$. The latter case occurs if we had received $uid_i = \perp$. In that case, we will extract z_i from π_{nym}, add z_i to our list of dummy identifiers $\mathcal{L}_{\mathsf{dummy}}$ and set $uid_i \leftarrow (\text{“rid“}||z_i)$ for the following procedure.

If it is the first time that sim sees the uid_i (either dummy, or real), it now creates the pseudonym records for all corrupt servers. More precisely, for each corrupt $\mathcal{S}_B \in \mathbf{S}$, sim sends $(\mathsf{NYMGEN}, sid, uid_i, \mathcal{S}_B, 1)$ to $\mathcal{F}$, and subsequently provides the pseudonym value $nym^*_{i,B}$ by sending $(\mathsf{NYMGEN}, sid, \mathcal{S}_B, nym^*_{i,B})$ to $\mathcal{F}$. Thereby, $nym^*_{i,B}$ is computed as $nym^*_{i,B} \leftarrow z_i^{x_B}$ where z_i is either taken from $uid_i = (\text{“rid“}||z_i)$ or computed as $z_i \leftarrow \mathsf{PRF}_{\mathbb{G}}(x_{\mathcal{X}}, uid_i)$ if $uid_i \neq (\text{“rid“}||z_i)$. That is, in the simulation we register the inner pseudonym values $xnym_{i,B}$ for all corrupt servers as pseudonyms in the ideal functionality.

Pseudonym Conversion. Here the main part is done by the corrupt converter, thus in the simulation we mainly have to ensure that an honest server can always send a correct looking request to $\mathcal{X}$ and also reflect the conversions done by the adversary in the ideal functionality.

from honest to honest server: We start the simulation when sim receives $(\mathsf{CONVERT}, sid, qid, \mathcal{S}_A, \mathcal{S}_B)$ from the ideal functionality. First, "$\mathcal{S}_A$" computes $C \xleftarrow{\$} \mathsf{Enc}_{\mathbb{G}}(epk_{\mathcal{X}}, \mathsf{Enc}_{\mathbb{G}}(epk_B, 1))$ and $\sigma_C \leftarrow \mathsf{Sign}(ssk_A, (sid, qid, C))$ and fakes the corresponding proof π_A. "$\mathcal{S}_A$" then sends the request $(sid, qid, C, \pi_A, \sigma_C, \mathcal{S}_B)$ via $\mathcal{F}_{\mathrm{SMT}}$ to $\mathcal{X}$. When "$\mathcal{S}_B$" receives a valid response $(sid, qid, C, C'', \sigma_C, \overline{\sigma}_{nym}\pi_A, \pi_{\mathcal{X}}, S_A)$ from $\mathcal{X}$, i.e., where $\pi_{\mathcal{X}}$ is correct and π_A, σ_C are the values sent by "$\mathcal{S}_A$", sim sends $(\mathsf{PROCEED}, sid, qid)$ to $\mathcal{F}$ and subsequently triggers the delivery of the converted pseudonym to the ideal world server $\mathcal{S}_B$ as well.

from corrupt to honest server: There is nothing to simulate in the real world here, we only mimic the observed behaviour in the ideal world. First, when "$\mathcal{S}_B$" in the real word receives a valid tuple $(sid, qid, C, C'', \sigma_C, \overline{\sigma}_{nym}, \pi_A, \pi_{\mathcal{X}}, S_A)$, i.e., where the signature and both proofs are correct, it extracts $xnym_{i,A}$ from π_A. If sim has not seen $xnym_{i,A}$ before, it quickly generates pseudonym records based on $xnym_{i,A}, \mathcal{S}_A$ in $\mathcal{F}$, before starting the conversion request. This is not done only for $\mathcal{S}_A$ but for *all* corrupt servers again. Thereby, sim does not know the underlying uid_i, though. However, we use a similar procedure as in the pseudonym generation described above and link the pseudonyms to a dummy identifier $(\text{“rid“}||z_i)$ instead. Therefore, we first compute $z_i \leftarrow xnym_{i,A}^{1/x_A}$ and add z_i to $\mathcal{L}_{\mathsf{dummy}}$. Then, for all corrupt servers $\mathcal{S}_C \in \mathbf{S}$ (incl. $\mathcal{S}_A$) we send the message $(\mathsf{NYMGEN}, sid, (\text{“rid“}||z_i), \mathcal{S}_C, 1)$ to $\mathcal{F}$, followed by a call $(\mathsf{NYMGEN}, sid, \mathcal{S}_C, nym^*_{i,C})$ where $nym^*_{i,C} \leftarrow z_i^{x_C}$.

When this is done, we trigger a conversion request in the ideal world by sending $(\mathsf{CONVERT}, sid, qid, xnym_{i,A}, \mathcal{S}_B)$ to $\mathcal{F}$ followed by the input $(\mathsf{PROCEED}, sid, qid)$. This will result in the message $(\mathsf{CONVERTED}, sid, qid, \mathcal{S}_A, nym_{i,B})$ being delivered to the ideal world server $\mathcal{S}_B$ where $nym_{i,B}$ is a random pseudonym value.

As in the pseudonym generation described above we have ensured that whenever an honest $\mathcal{S}_B$ receives a new pseudonym $nym_{i,B}$, all related pseudonyms for corrupted servers are defined within $\mathcal{F}$ too.

from honest to corrupt server: Here sim starts when it receives a message (CONVERT, $sid, qid, \mathcal{S}_A, \mathcal{S}_B$) from $\mathcal{F}$. It then completes the pseudonym request towards the ideal functionality to learn the converted pseudonym $nym_{i,B}$ of the corrupt server $\mathcal{S}_B$. Thus, sim sends (PROCEED, sid, qid) and (PROCEED, $sid, qid, \perp$) to $\mathcal{F}$. That is, the simulator doesn't actually specify the pseudonym $nym^*_{i,B}$ (which it could, since $\mathcal{S}_B$ and $\mathcal{X}$ are corrupt). However, this input would have been ignored anyway since we will have already ensured that $nym_{i,B}$ is defined within $\mathcal{F}$. Recall, that we have generated related pseudonym records for *all* corrupt servers, whenever a pseudonym for an honest server was created. Thus, from the response (CONVERTED, $sid, qid, \mathcal{S}_A, nym_{i,B}$) that sim gets from $\mathcal{F}$, we can now determine the corresponding inner pseudonym $xnym_{i,A}$ and start a correct conversion in the real world. Namely, as we have generated $nym_{i,B}$ using the converter's extracted keys, we can now compute $xnym_{i,A} \leftarrow nym_{i,B}^{x_A/x_B}$. From that point, we now run the normal procedure (according to GAME 11) for "$\mathcal{S}_A$", meaning that C and σ_C are generated honestly from $xnym_{i,A}$ whereas the proof π_A is simulated.

Thus, we have shown how to construct a simulator that provides a view that is indistinguishable to the one described in GAME 12, which concludes our proof.

■