With a Little Help from My Friends: Constructing Practical Anonymous Credentials

Lucjan Hanzlik
lucjan.hanzlik@cispa.de
CISPA Helmholtz Center for Information Security
Saarbrücken, Germany

Daniel Slamanig
daniel.slamanig@ait.ac.at
AIT Austrian Institute of Technology
Vienna, Austria

ABSTRACT

Anonymous credentials (ACs) are a powerful cryptographic tool for the secure use of digital services, when simultaneously aiming for strong privacy guarantees of users combined with strong authentication guarantees for providers of services. They allow users to selectively prove possession of attributes encoded in a credential without revealing any other meaningful information about themselves. While there is a significant body of research on AC systems, modern use-cases of ACs such as mobile applications come with various requirements not sufficiently considered so far. These include preventing the sharing of credentials and coping with resource constraints of the platforms (e.g., smart cards such as SIM cards in smartphones). Such aspects are typically out of scope of AC constructions, and, thus AC systems that can be considered entirely practical have been elusive so far.

In this paper we address this problem by introducing and formalizing the notion of core/helper anonymous credentials (CHAC). The model considers a constrained core device (e.g., a SIM card) and a powerful helper device (e.g., a smartphone). The key idea is that the core device performs operations that do not depend on the size of the credential or the number of attributes, but at the same time the helper device is unable to use the credential without its help. We present a provably secure generic construction of CHACs using a combination of signatures with flexible public keys (SFPK) and the novel notion of aggregatable attribute-based equivalence class signatures (AAEQ) along with a concrete instantiation. The key characteristics of our scheme are that the size of showing tokens is independent of the number of attributes in the credential(s) and that the core device only needs to compute a single elliptic curve scalar multiplication, regardless of the number of attributes. We confirm the practical efficiency of our CHACs with an implementation of our scheme on a Multos smart card as the core and an Android smartphone as the helper device. A credential showing requires less than 500 ms on the smart card and around 200 ms on the smartphone (even for a credential with 1000 attributes).

CCS '21, November 15–19, 2021, Virtual Event, Republic of Korea.
© 2021 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-8454-4/21/11…$15.00
https://doi.org/10.1145/3460120.3484582

CCS CONCEPTS

• **Security and privacy → Digital signatures**; **Hardware-based security protocols**; • **Theory of computation → Cryptographic primitives**.

KEYWORDS

Anonymous credentials; secure elements; smart cards; mobile;

ACM Reference Format:
Lucjan Hanzlik and Daniel Slamanig. 2021. With a Little Help from My Friends: Constructing Practical Anonymous Credentials. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21), November 15–19, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 20 pages. https://doi.org/10.1145/3460120.3484582

1 INTRODUCTION

Anonymous credential systems (ACs), envisioned by Chaum [33] in the 1980ies and meanwhile found as commercial products such as U-Prove [65] or Idemix [30], allow users to obtain digital credentials from an issuer and to prove possession of attributes encoded in a credential, e.g., just prove that the holder is over 21 years old, to verifiers without revealing any other meaningful information about themselves. Typically, a credential contains a number of attributes, e.g. a collection of attributes such as age, address, gender, etc. for human credential holders or a potentially large number of attributes describing a platform and its configuration, e.g., for remote attestation.[1] These attributes can be selectively shown and thus support minimum disclosure, i.e., only information that is required for the particular application is revealed. The reason why ACs are considered useful is because they provide strong authentication and in addition strong privacy. This means that verifiers can be convinced that users really hold credentials from an issuer when the authentication is successful, but at the same time the credential issuer and verifiers (even if they collaborate) cannot link credentials to a specific session with the user.

There are two variants of ACs, namely *one-show* and *multi-show*. If ACs are *one-show* private, with U-Prove [65] being the most well known representative, then each credential can only be used once in an unlinkable way (i.e., multiple showings can be linked). While this might pose serious limitations in some settings, it has recently been found real-world applications and in particular in the form of PrivacyPass [41] by Cloudflare (available as extensions for Chrome and Firefox), the enhanced variant by Google [56] being integrated

[1]Remote attestation allows a verifier to determine a level of trust in the integrity of the platform of another system, i.e., the machine that holds the credential.

into the Trust Tokens API[2] or the PrivateStats proposal by Facebook.[3] A stronger variant of ACs is called *multi-show* private, which additionally guarantees that the repeated use of the same credential is unlinkable. The latter is a much more general and typically more desirable notion and we are exclusively focusing on multi-show ACs in this paper.[4] Multi-show ACs have a variety of applications such as access control to online-service [68], anonymous subscriptions [12, 57], e-tickets [50, 61] or point collection systems [14, 15]. A recent large scale real-world application of such ACs is the realization of private groups within the popular Signal messenger [32]. Moreover, there are recent innovative proposals such as Gradient's identity management infrastructure[5] supporting provable statements and claims chained to immutable (hardware-based) roots of trust via recent AC constructions [38, 39, 45].

Camenisch and Lysyanskaya [28] were the first to fully construct this cryptographic primitive. Their scheme is based on so-called CL-signatures that use RSA groups and allow to efficiently prove knowledge of a signature. In their follow-up work [29] they construct CL-signatures from bilinear groups and more schemes follow their template, e.g., [58, 66]. Brands [18] proposed an alternative construction (later made provably secure in [3]) that uses pairing-free groups at the expense of multi-show privacy. Besides the already mentioned constructions of ACs, there is significant research into different approaches to construct AC systems with various trade-offs in bandwidth, computational efficiency and security (e.g., [3, 27, 42, 45, 48, 69]). We will compare our approach to the most important ones later. Furthermore, there are various variants of ACs such as keyed-verification [31, 37], updatable [14, 36], delegatable [7, 13, 38], decentralized [46, 73] or cloud-based ACs [55], further broadening the scope of potential applications.

Preventing unauthorized sharing of credentials. The use of ACs in commercial products such as U-Prove or Idemix created new problems such as the sharing of credentials, allowing for instance non-paying or non-authorized users to gain access to a service (e.g., watch R-rated movies). A simple solution is to store the credential inside a secure hardware device (secure element) such as a smart card, which makes sharing a credential practically infeasible. This not only solves the problem of dishonest users, but provides an additional layer of security for credentials of honest users. It also allows applying ACs in e-government applications [11], since electronic identities (e-IDs) are usually based on smart cards. The problem that one encounters here, however, is that the AC constructions mentioned before are not designed having this in mind. Thus their efficiency is only practical on rather powerful devices such as PCs or smartphones, but fails on constrained devices such as a smart card providing much less memory and processing capabilities. Thus, they are typically far too inefficient for the use in such a setting.

Anonymous credentials on constrained devices. There were several attempts to implement ACs on smart cards. Bichsel et. al. [10] implemented CL credentials [28] on a standard Java Card [64]. Unfortunately, for a meaningful security parameter, their implementation required more than 16 seconds to perform a showing. A more practical implementation was proposed by Mostowski and Vullers [63]. They implemented U-Prove like one-show ACs on a smart card in Multos technology [60], where proving possession of 1 of 5 attributes in a credential takes around 0.9s (Bjones et al. [11] report about 0.5s for 10 undisclosed attributes). Recently Camenisch et. al. in [23] proposed a construction and smart card implementation of keyed-verification ACs [31], a restricted class of ACs where the issuer is also the verifier. They achieve execution times similar to the aforementioned one in [63]. A somewhat different approach to ACs was proposed by Batina et. al. [6]. Here, a credential is associated with a randomizable certificate on the user's public key (which can also be randomized). Therefore, each credential corresponds to a single attribute. For a showing, the user randomizes the public key, the certificate, and signs a nonce send by the verifier. The concrete construction uses self-blindable certificates by Verheul [74] and their implementation requires around 3s to show one credential/attribute at a 100 bit security level.

Drawbacks of existing implementations. The main drawback of all these implementations is that the execution time on the smart card depends on the number of attributes and either increases with the number of disclosed or undisclosed attributes but always linearly increases with the number of attributes inside the credential. Due to this reason, the application of smart card based ACs is limited to cases where the user possess only a very small number of attributes and very soon gets impractical in use-cases that require more attributes. For smart cards, Mostowski and Vullers in [63] report that adding an attribute to the credential increases the execution time of a showing by around 0.1s. We stress that while in case of PC or smartphone implementations one still notices the linear increase in execution time, it is significantly less problematic than in case of smart cards.

On the number of attributes. Attributes provided by governmental issuers usually reflect basic personal information about the credential holder (e.g. name, gender, age, address). However, there are many scenarios where additional attributes can be defined. In particular, the IRMA pilot implementation of AC's developed by the Privacy by Design Foundation[6] provided several real-world attributes considered by the industry/government like diplomas, certificates, or even membership IDs for online services (e.g. Facebook ID). Moreover, in the context of eIDs in some European countries, e.g., Austria or Germany, service-specific pseudonyms are used for authentication and computing them on the fly would be too expensive. Therefore a more efficient approach would be to store them as attributes inside the credential. It is worth noting that in Austria according to [54] there are around 30 of them for governmental purposes and potentially many more for other industrial purposes. Attributes however can not only be used to describe individuals but are also useful to reflect properties of the user's platform or other devices like servers. For example, when basing access control on the configuration of the platform, one can consider binary attributes such as whether a certain software, e.g. antivirus, or some hardware, e.g. certain sensor type, is present. Note there could be

[2]https://web.dev/trust-tokens/

[3]https://research.fb.com/privatestats

[4]Note that every multi-show AC can easily be turned into a one-show AC by including a unique attribute that always needs to be shown.

[5]https://www.gradient.tech/

[6]https://privacybydesign.foundation/attribute-index/en/

numerous such attributes and in addition those properties could also be arbitrarily valued, e.g., OS type, version, hardware vendor. In such a case the number of attributes in the system is likely to be large.

As efficiency of the system is influenced by the number of attributes in the credentials, this aspect gets even more important considering examples like the ones above where the number of potential attributes in some scenarios can be in the tens to hundreds.

Our goals and setting. To overcome the aforementioned problems, we consider splitting the overall computations between a resource constrained device, e.g., a secure element (SE) such as a smart card in Figure 1 (*the core*), and a much more powerful host device, e.g., a primary device such as a smartphone in Figure 1 (*the helper*). And in particular our goal is to consider this *core/helper* setting already in the formal AC model. The motivation comes from the observation that nowadays platforms that use ACs (e.g., PCs, smartphones) typically are equipped with secure elements (SEs) in form of dedicated hardware modules, e.g., the Trusted Platform Module (TPM)[7] or SIM cards that are designed to handle secrets (such as secret keys for ACs). Besides, many modern processors come with hardware-enforced isolation that is already built into the CPU and allows to build trusted execution environments (TEE), e.g., TrustZone by Arm or the Software Guard Extensions (SGX) by Intel. Such TEEs feature isolated execution of user processes and are also used to emulate TPM functionality [67] (e.g., Intel fTPM). Since there is a huge body on recent practical microarchitectural attacks on TEEs, this however questions their adequacy for cryptographic applications (cf. [40, 70]). Consequently, we focus on hardware SEs such as TPMs or SIM cards more suitable for handling cryptographic keys.[8] Nevertheless, insights from an implementation and its performance on such constrained SEs gives us a good baseline, as performance will only get better if we move to "software-based" TEEs like TrustZone or SGX.

Now, any such SE (*core* device) depends on a host device (*the helper*) that provides power supply and acts as a gateway to the outside world. Besides TPMs and SIM cards in PCs and smartphones, this is also true for the Internet of Things (IoT), where smaller and constrained devices are connected to a more powerful IoT hub. In most applications, the used helper device is owned by the user and can be leveraged to perform part of the computation and can also be used to store larger amounts of data. So while we consider the helper to be potentially malicious, a well known problem in such a setting is that a corrupted helper device can always break the privacy of an AC system, e.g., by adding identifying metadata before finalizing the showing with a verifier. This can obviously not be checked by the core device. But we can take advantage of this fact and prioritize the efficiency of the core at the expense of protecting privacy against the helper.[9] Nevertheless, we do not want to tolerate that a malicious helper can show a credential without interacting with the core. Consequently, we require that as long as the verifier sends an honest challenge triggering the showing of a credential at the helper, the core needs to be involved in order to result in a valid showing of the credential and even a malicious helper cannot succeed.

High-level overview of our CHAC approach. We are now ready to provide a high-level overview of our core/helper anonymous credentials (CHAC) approach (cf. Figure 1). Initially, the core generates a secret key ① which never leaves the core; the user can now obtain multiple credentials from an issuer by ② sending a request, which is then ③ passed to the core (ensuring that core needs to be involved in obtaining credentials) and after the issuing ④ is finished, the credentials are stored at the helper ⑤. For a showing, the helper first triggers a request ⑥, which is then passed to the core ⑦ (again ensuring that core needs to be involved). Then, depending on the attributes that need to be selectively shown (all other remain undisclosed) the helper can aggregate them from potentially different credentials into a compact showing token ⑧. Note that while for certain applications (e.g., the core being a SIM card in the smartphone) batching may not be so important, but if we for instance consider a standalone NFC based smart card, the communication between the core and the smartphone is limited because of the way the user has to physically interface both devices. Therefore some kind of batching (aggregation) is desirable, i.e., the helper device should be able to accumulate many showing tokens for the core into a single compact one. Finally, the helper sends the resulting showing token to the verifier ⑨ who either accepts or rejects. Showings can be performed with different verifiers and an arbitrary number of times without the showings being linkable to each other.

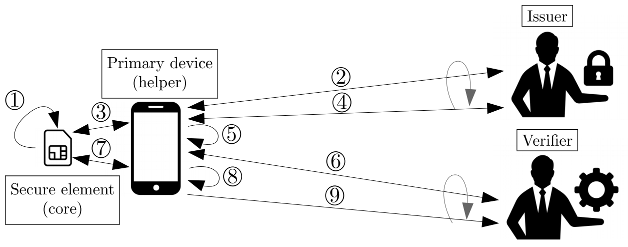

Figure 1: High-level overview of our approach.

Previous work in the core/helper setting. In order to put our CHAC approach into context, we will look at one well known example for the core/helper setting. Namely, the direct anonymous attestation (DAA) protocol [20, 22] designed for privacy-preserving remote attestation of platforms. Here the core device is the Trusted Platform Module (TPM), a specialized chip supporting DAA, and the helper is a PC. Technically, DAA is not an AC system, but rather a group signature scheme [34] (without the anonymity revocation capability), but with a mechanism to detect rogue members and optional linkability. It can be considered as the most widely deployed protocol for anonymous authentication in practice[10]. Previously, there have been informal discussions on how a TPM can be used together with CL-credentials in [21] as well as explicit constructions

[7] https://trustedcomputinggroup.org/resource/tpm-library-specification/

[8] Although it clearly needs to be mentioned that these are not immune against attacks (cf. [62] for recent timing side-channels in TPMs.)

[9] Note that this can never be prevented by the core device and in practice it is more likely that malware running on the helper device will use this approach to leak private information about the user than breaking the actual cryptographic scheme.

[10] An enhanced DAA with revocation capabilities is called Enhanced Privacy ID (EPID) [19] and revocation was later also adopted for existing DAA [22, 25]. EPID, however, is not designed for the core/helper setting.

Scheme	Show (Core/Helper)	Verify	$\|\text{Cred}\|$	$\|\text{Show}\|$
[25] (DAA-A)	$3\mathbb{G}_1$ / $O(U\mathbb{G}_1)$	$O(L\mathbb{G}_1) + 2P$	$2\mathbb{Z}_p + 2\mathbb{G}_1$	$O(U\mathbb{Z}_p) + 4\mathbb{G}_1$
[22] (DAA-A)	$3\mathbb{G}_1$ / $O(U\mathbb{G}_1)$	$O(L\mathbb{G}_1) + 2P$	$2\mathbb{Z}_p + 2\mathbb{G}_1$	$O(U\mathbb{Z}_p) + 4\mathbb{G}_1$
[26] (DAA-A)[a]	$3\mathbb{G}_1$ / $O(L\mathbb{G}_1)$	$O(L\mathbb{G}_1) + 4P$	$O(L\mathbb{G}_1)$	$O(L(\mathbb{G}_1 + \mathbb{Z}_p))$
CHAC	$1\mathbb{G}_1$ / $O(D(\mathbb{G}_1 + \mathbb{G}_2))$	$O(DP)$	$O(L(\mathbb{G}_1 + \mathbb{G}_2))$	$6\mathbb{G}_1 + 3\mathbb{G}_2$

[a] This LRSW based DAA scheme is supported in FIDO. Though it does not support attributes, for completeness we include a projection of its complexity if realized as DAA-A based on the LRSW based DAA-A in [35].

Table 1: Comparison of CHAC with existing DAA-A constructions. $|\cdot|$ denotes sizes and otherwise computational effort. For Type-3 pairings and the BN-256 curve we have in bits $|\mathbb{G}_2| = 2 \cdot |\mathbb{G}_1|$, $|\mathbb{G}_1| = 2 \cdot |\mathbb{Z}_p|$, and $|\mathbb{Z}_p| = 256$.

that extend DAA with attributes (DAA-A) and selective attribute disclosure [22, 25, 35], bringing it closer to AC systems. DAA(-A) constructions however are proven secure in a formal model that exactly captures DAA(-A), with a long line of failed security notions [22, 26], and a design tailored towards a specific core device being the TMP (2.0). With CHAC, our aim is to have a simpler and much more general model not tailored to a specific core device. We note that CHAC can be an alternative to DAA in some of its use-cases, but due to DAA(-A)'s focus on specific features, e.g., linkability, it is not intended to be a replacement.

Since all aforementioned DAA constructions follow the same template, they all have the same inherent performance drawbacks. In Table 1 we compare our CHAC construction to the recent DAA-A proposals, where we denote k exponentiations in group $\mathbb{G}_i$ in a bilinear group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, p)$ with pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ by $k\mathbb{G}_i$ and kP denotes k pairing operations. Moreover, we denote by L the number of attributes and by D and U the number of selectively disclosed and undisclosed attributes respectively. We see that CHAC asymptotically improves over DAA-A and concretely we improve significantly on the core (the most critical part) and size of the showing token. For practical applications, where one can assume that $D \ll U$ as this is the main use-case of a selective disclosure tool for privacy, we also improve significantly (cf. Section 5 for a detailed discussion). We note that while our credentials are larger compared to other work, they are stored on the helper device where storage space is not an issue. Moreover, for practical numbers of attributes the credentials are still relatively small, i.e., around 200KB for 100 attributes.

Scheme	$\|\text{Params}\|$	Show	Verify	$\|\text{Cred}\|$	$\|\text{Show}\|$
[45, 48]	$O(L)$	$O(U)$	$O(D)$	$O(1)$	$O(1)$
[27]	$O(L)$	$O(U)$	$O(D)$	$O(1)$	$O(1)$
[69]	$O(L^2)$	$O(U)$	$O(D)$	$O(1)$	$O(1)$
[49]	$O(L)$	$O(1)$	$O(D)$	$O(L)$	$O(1)$
CHAC	$O(L)$	$O(D)$	$O(D)$	$O(L)$	$O(1)$

Table 2: Comparison of CHAC (merging core and helper) with conventional ACs designed for selective disclosure.

Comparing core/helper ACs to conventional ACs. Finally, for the sake of completeness we want to put our CHAC approach into context of existing conventional state-of-the-art AC systems that *do not* consider this core and helper separation. We focus on ACs that like our approach provide constant-size selective showing of attributes [27, 45, 48, 49, 69, 73]. Since this is not our main focus of the paper, in Table 2 we only provide an asymptotic comparison of the characteristics when using our CHAC approach as a conventional AC system by merging the core and helper functionality into a single entity. A rough comparison based on expensive operations, i.e., group exponentiations and pairings,[11] and for fairness assuming that $D = U < L$ yields that for [45, 48] showing and verification are equivalent. [69] has comparable verification efficiency but less efficient showings. In the recent concurrent and independent work in [49], which also uses an aggregatable approach as in our construction, verification is equivalent, but their showing is more efficient and requires only a constant number of expensive operations. Finally, the showing of the most compact scheme from [27] includes around 100 group elements and the computational costs are not even evaluated, but can be assumed too high in practice (especially for constrained devices).

Note, however, that vice versa it is not straightforwardly possible for the other AC approaches to achieve our core/helper separation. As can be seen, while our CHAC approach has larger credentials, which as discussed above is not really an issue, we outperform all existing approaches in that the computation within showing and verification is in the number D of disclosed attributes, a number that is typically very small compared to U and L in practical privacy-preserving applications. Consequently, our CHAC approach also yields an interesting alternative when not requiring this core/helper separation.

1.1 Our Contribution and Technical Overview

Our contributions can be summarized in points as follows:

Formal framework for CHAC. We formalize a cryptographic primitive called core/helper anonymous credentials (CHAC). The key idea is that the core device performs operations that do not depend on the size of the credential or the number of attributes. While we cannot guarantee privacy in front of a malicious helper device, we however require that even a malicious helper device is not able to perform a credential showing without the help of the core. In particular, after n showings by the core, even a malicious helper is not able to produce more than n valid showings. We call the later property *dependability*. Besides the usual *unforgeability* and *anonymity*, which are defined similarly to previous work on ACs, we also consider a property called *compactness*. It states that the size of showing of a credential (called show token) should be independent of the number of disclosed/undisclosed attributes.

Generic construction. We provide a construction of CHACs inspired by the approach to construct single-attribute credentials

[11] A comparison based on implementations would be very interesting, but for most schemes no open implementations are available.

from self-blindable certificates [6]. However, instead of using Verheul's scheme [74], we instantiate self-blindable credentials using the approach by Backes et. al. [1]. They introduced signatures with flexible public keys (SFPK) and showed that they can be efficiently combined with signatures on equivalence classes (SPS-EQ) [43, 45, 48, 53, 59]. In brief, SFPK are signatures where the *key space* is partitioned into equivalence classes and a signer can efficiently change a key pair to a different representative of the same class that is indistinguishable from a newly generated one. SPS-EQ are signatures where the *message space* is partitioned into equivalence classes and everyone can update a signature to another representative of the message class, where the resulting signature is indistinguishable from a fresh one. We will usually denote this update operation (the change of representative) by *adapt*.

The starting point for our generic construction is to represent a credential as a SPS-EQ signature on a SFPK public key and the core device just generates a SFPK signature. The helper device adapts the SFPK public key, randomizes the SFPK signature and adapts the SPS-EQ signature to the updated SFPK public key. Unfortunately, similar to [6], this only yields a single-attribute credential. To overcome this limitation, we build upon the notion of SPS-EQ and introduce two cryptographic primitives that are of independent interest: tag-based equivalence class signatures (TBEQ) and aggregatable attribute-based equivalence class signatures (AAEQ). In contrary to standard equivalence class signatures, TBEQ allow to additionally include a tag (an attribute value) when signing a message (class). AAEQ then allow to aggregate multiple TBEQ signatures under different keys (representing attributes) and tags (representing attribute values) on the same message (representative). In our construction, we then use AAEQ instead of SPS-EQ in the above template, which allows us to aggregate multiple certificates to different attributes and attribute values into a single one. In other words, during the show procedure the helper device randomizes the SFPK signature and adapts the public key, chooses the certificates corresponding to the disclosed attributes, aggregates them into a single compact AAEQ signature and adapts it to the updated SFPK public key. The core device still only generates the SFPK signature and thus the helper device is unable to use the credential without a valid SFPK signature from the core device.

Efficient CHAC instantiation. We instantiate the construction described above using schemes that are secure in the generic group model [72] and in addition use random oracles [8]. We note that both are idealized assumptions and it would be more favorable to have a scheme secure only in the ROM or even in the standard model. Unfortunately, we do not yet have building blocks available that are efficient and do not require such assumptions. As our main motivation is a highly practical solution, we opted for efficiency at the cost of idealized assumptions.

Our SFPK signature builds upon the one by Backes et. al. [2], but we replace the programmable Waters hash function [75] with a random oracle. We instantiate our primitives in Type-3 bilinear groups $\mathsf{BG} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, p)$ using the popular BN-256 curve [4] and the optimal ate pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. The signing process involves operations in $\mathbb{G}_2 = E(\mathbb{F}_{p^2})$ which are not natively supported by smart cards and should be avoided. Therefore, we show how to securely split the signing process into three steps: a pre-computation step that is performed only once, the main part that only involves operations in $\mathbb{G}_1$, natively supported by smart cards, and a finalization step that can be performed without the secret key. This allows for the core device to pre-compute certain data once and then only sign using operations in $\mathbb{G}_1$ where the helper device will finalize the SFPK signature and perform operations in $\mathbb{G}_2$. We call this extension SFPK *with split signing*.

Our tag-based equivalence class signature (TBEQ) is based upon the SPS-EQ scheme from [44] extended with one component representing a one-time BLS signature [16] on the tag in group $\mathbb{G}_2$ using the randomness of the SPS-EQ scheme as a one-time signing key. The corresponding verification key is already part of the SPS-EQ scheme from [44]. Similar to [44] we analyze its security in the generic group model. In order to construct a provably secure aggregatable attribute-based equivalence class (AAEQ) scheme, we use parallel copies of this TBEQ scheme with independent keys, where all instances compute the signing randomness deterministically using a PRF evaluation on the message using a shared PRF key. We again prove it secure in the generic group model.

Efficient CHAC implementation. We provide an efficient prototype implementation that uses a Multos smart card as the core device and a smartphone with a Snapdragon 710 processor and 6GB RAM running Android 10.0 to implement the helper device and verification algorithm. For a comprehensive evaluation, we execute the same code on a PC (laptop) with Intel i7-7660U CPU @ 2.50 GHz with 16GB RAM. The execution time on the core device with the BN-256 curve (providing around 100-bit of security) is $<$ 0.5s. The helper device part for credentials even with 1000 attributes takes $\approx$ 200ms for the smartphone and 15ms for the PC which respectively adds to 0.7s and 0.5s for a full showing of 1000 attributes. Verification of such a show token takes $\approx$ 800ms on the PC and $\approx$ 100ms if we assume that the verifier knows the set of potential attribute/value pairs and does some pre-computation. For show tokens with 10 and 100 attributes, the verification takes respectively 140ms and 200ms even without this optimization. The most computationally expensive operation is the issuing which takes $\approx$ 200ms and $\approx$ 1s for credentials with 10 and 100 attributes respectively. However, we show that issuing can be distributed and the workload decreases with the number of used cores/servers.

Extensions and Optimization. Finally, we discuss various extensions and optimizations of our CHAC instantiation.

2 PRELIMINARIES

We denote by $y \xleftarrow{\$} \mathcal{A}(x)$ the execution of algorithm $\mathcal{A}$ on input x and with output y. By $r \xleftarrow{\$} S$ we mean that r is chosen uniformly at random from set S. We will use $1_{\mathbb{G}}$ to denote the identity element in group $\mathbb{G}$ and $[n]$ to denote the set $\{1, \ldots, n\}$. We will denote a bilinear group as $\mathsf{BG} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, p)$ and will consider Type-3 pairings, i.e., there is no efficiently computable isomorphism between $\mathbb{G}_1$ and $\mathbb{G}_2$. Finally, by $\mathcal{A}^{O}$ we denote an algorithm $\mathcal{A}$ that has access to oracle O. We defer some further notation that is not required in the main body of the paper to Appendix A.1.

2.1 Signatures on Equivalence Classes

Structure-preserving signatures on equivalence classes (SPS-EQ) [45, 48] sign vectors of length $\ell > 1$ from one of the prime order p source groups $\mathbb{G}_i$ ($i \in \{1, 2\}$) of a bilinear group BG. We can view $\mathbb{Z}_p^\ell$ as a vector space and one can define a projective equivalence relation on it, which propagates to $\mathbb{G}_i^\ell$ and partitions $\mathbb{G}_i^\ell$ into equivalence classes. An SPS-EQ-scheme signs equivalence classes $[M]$ of vectors $M \in (\mathbb{G}_i^*)^\ell$ with equivalence relation: $M, N \in \mathbb{G}_i^\ell : M \sim_{\mathcal{R}} N \Leftrightarrow \exists s \in \mathbb{Z}_p^* : M = N^s$, i.e., scaling the message by s.

Definition 2.1 (SPS-EQ). An SPS-EQ scheme SPS-EQ on message space $(\mathbb{G}_i^*)$ for $i \in \{1, 2\}$ consists of the following PPT algorithms.

$\mathsf{Setup}(1^\lambda)$**:** on input a security parameter 1^λ, outputs group BG.

$\mathsf{KeyGen}(\mathsf{BG}, \ell)$**:** on input BG and message vector length $\ell > 1$, outputs a key pair $(\mathsf{pk}, \mathsf{sk})$.

$\mathsf{Sign}(\mathsf{sk}, M)$**:** on input a secret key sk and representative $M \in (\mathbb{G}_i^*)^\ell$, outputs a signature σ for equivalence class $[M]$.

$\mathsf{ChgRep}(M, \sigma, \mu, \mathsf{pk})$**:** on input representative $M \in (\mathbb{G}_i^*)^\ell$ of equivalence class $[M]$, a signature σ on M, a value μ and a public key pk, returns an updated message-signature pair (M', σ'), where the new representative is $M' = M^\mu$ and σ' its corresponding (or, updated) signature.

$\mathsf{Verify}(\mathsf{pk}, M, \sigma)$**:** is a deterministic algorithm and, on input a public key pk, a representative $M \in (\mathbb{G}_i^*)^\ell$, and a signature σ outputs a bit $b \in \{0, 1\}$.

$\mathsf{VKey}(\mathsf{sk}, \mathsf{pk})$**:** is a deterministic algorithm and, on input secret key sk and a public key pk, checks if it represents a valid key pair and outputs a bit $b \in \{0, 1\}$.

We provide formal definitions of security in Appendix A.3.

2.2 Signatures with Flexible Public Key

Signatures with flexible public key (SFPK) [1] are signatures that provide relations $[\mathsf{pk}]_{\mathcal{R}}$ on public keys. The main property is called class-hiding and states that it is hard to decide if a random public key is in a relation to a different public key. We use the class-hiding definition with key corruption introduced in [2], where the adversary gets the secret keys. This definition is weaker than in [1], but allows to instantiate this primitive with a shorter (and optimal) public key of 2 group elements, as shown in [2].

*Definition 2.2 (*SFPK*).* A SFPK scheme is a set of *PPT* algorithms such that:

$\mathsf{SFPK.CRSGen}(1^\lambda)$**:** on input a security parameter 1^λ, outputs a trapdoor δ_ρ and a common reference string ρ, which is an implicit input for all the algorithms.

$\mathsf{SFPK.KeyGen}(1^\lambda)$**:** on input a security parameter 1^λ outputs a key pair $(\mathsf{sk}, \mathsf{pk})$.

$\mathsf{SFPK.TKGen}(1^\lambda)$**:** on input a security parameter 1^λ outputs a key pair $(\mathsf{sk}, \mathsf{pk})$, and a trapdoor δ.

$\mathsf{SFPK.Sign}(\mathsf{sk}, m)$**:** on input a message $m \in \{0, 1\}^*$ and a signing key sk, outputs a signature Sig.

$\mathsf{SFPK.ChkRep}(\delta, \mathsf{pk}')$**:** on input a trapdoor δ for some equivalence class $[\mathsf{pk}]_{\mathcal{R}}$ and public key pk', outputs 1 if $\mathsf{pk}' \in [\mathsf{pk}]_{\mathcal{R}}$ and 0 otherwise.

$\mathsf{SFPK.ChgPK}(\mathsf{pk}, r)$**:** on input a representative pk of equivalence class $[\mathsf{pk}]_{\mathcal{R}}$ and random coins r, outputs a different representative pk', where $\mathsf{pk}' \in [\mathsf{pk}]_{\mathcal{R}}$.

$\mathsf{SFPK.ChgSK}(\mathsf{sk}, r)$**:** on input a secret key sk and random coins r, outputs an updated secret key sk'.

$\mathsf{SFPK.Verify}(\mathsf{pk}, m, \mathsf{Sig})$**:** on input a message m, signature Sig and public verification key pk, outputs 1 if the signature is valid and 0 otherwise.

Definition 2.3 (Canonical Representative). Let canon be a predicate that holds for exactly one public key in a given class. We say $\mathsf{pk}_{\mathsf{SFPK}}$ is a canonical representative if $\mathsf{canon}(\mathsf{pk}_{\mathsf{SFPK}}) = 1$.

We provide the formal security definitions in Appendix A.2.

3 NEW RESULTS AND BUILDING BLOCKS

In this section we provide new results on SFPK signatures and introduce tag-based equivalence class (TBEQ) signatures as well as aggregatable attribute-based equivalence class (AAEQ) signatures.

3.1 Efficient SFPK with Split Signing

We base our SFPK signature scheme on the one by Backes et al. [2], but we replace the programmable Waters hash function [75] with a hash function H modeled as a random oracle. This allows us to increase the efficiency of the signing process, i.e., we replace $O(\lambda)$ group operations in $\mathbb{G}_1$ with one hashing to $\mathbb{G}_1$. The change requires us to prove security in the random oracle model. However, it also allows us to securely divide the signing process so that in our CHAC the core only performs operations in $\mathbb{G}_1$ and can seek support by the helper device to finish the signing process without knowing the secret key.

$\underline{\mathsf{SFPK.CRSGen}(1^\lambda)}$: generate $\mathsf{BG} \xleftarrow{\$} \mathsf{BGGen}(\lambda)$, choose $y \xleftarrow{\$} \mathbb{Z}_p^*$ and compute $Y_1 = g_1^y$ and $Y_2 = g_2^y$. Set $\rho = (\mathsf{BG}, Y_1, Y_2)$.

$\underline{\mathsf{SFPK.KeyGen}(1^\lambda)}$: choose $x \xleftarrow{\$} \mathbb{Z}_p^*$. Set $\mathsf{pk}_{\mathsf{SFPK}} = (g_1, g_1^x)$ and $\mathsf{sk}_{\mathsf{SFPK}} = (Y_1^x, \mathsf{pk}_{\mathsf{SFPK}})$.

$\underline{\mathsf{SFPK.TKGen}(1^\lambda)}$: choose $x \xleftarrow{\$} \mathbb{Z}_p^*$. Set $\mathsf{pk}_{\mathsf{SFPK}} = (g_1, g_1^x)$, $\mathsf{sk}_{\mathsf{SFPK}} = (Y_1^x, \mathsf{pk}_{\mathsf{SFPK}})$, and $\delta_{\mathsf{SFPK}} = (g_2^x)$.

$\underline{\mathsf{SFPK.Sign}(\mathsf{sk}_{\mathsf{SFPK}}, m)}$: given a message $m \in \{0, 1\}^\lambda$, choose $r \xleftarrow{\$} \mathbb{Z}_p^*$ and return the signature $\mathsf{Sig}_{\mathsf{SFPK}} = (Y_1^x \cdot \mathsf{H}(m)^r, g_1^r, g_2^r)$.

$\underline{\mathsf{SFPK.ChgPK}(\mathsf{pk}_{\mathsf{SFPK}}, r)}$: Parse $\mathsf{pk}_{\mathsf{SFPK}} = (A, B)$ and compute $\mathsf{pk}'_{\mathsf{SFPK}} = (A^r, B^r)$. Return $\mathsf{pk}'_{\mathsf{SFPK}}$.

$\underline{\mathsf{SFPK.ChgSK}(\mathsf{sk}_{\mathsf{SFPK}}, r)}$: Parse $\mathsf{sk}_{\mathsf{SFPK}} = (Y_1^x, \mathsf{pk}_{\mathsf{SFPK}})$ and compute $\mathsf{pk}'_{\mathsf{SFPK}} \leftarrow \mathsf{SFPK.ChgPK}(\mathsf{pk}_{\mathsf{SFPK}}, r)$, and return $\mathsf{sk}'_{\mathsf{SFPK}} = ((Y_1^x)^r, \mathsf{pk}'_{\mathsf{SFPK}})$.

$\underline{\mathsf{SFPK.ChkRep}(\delta_{\mathsf{SFPK}}, \mathsf{pk}_{\mathsf{SFPK}})}$: $\mathsf{pk}_{\mathsf{SFPK}} = (A, B)$. Return 1 iff

$$e(A, \delta_{\mathsf{SFPK}}) = e(B, g_2).$$

$\underline{\mathsf{SFPK.Verify}(\mathsf{pk}_{\mathsf{SFPK}}, m, \mathsf{Sig}_{\mathsf{SFPK}})}$: parse $\mathsf{Sig}_{\mathsf{SFPK}}$ as $(\mathsf{Sig}^1_{\mathsf{SFPK}}, \mathsf{Sig}^2_{\mathsf{SFPK}}, \mathsf{Sig}^3_{\mathsf{SFPK}})$, parse $\mathsf{pk}_{\mathsf{SFPK}}$ as (A, B). Return 1 iff

$$e(\mathsf{Sig}^2_{\mathsf{SFPK}}, g_2) = e(g_1, \mathsf{Sig}^3_{\mathsf{SFPK}}) \text{ and}$$
$$e(\mathsf{Sig}^1_{\mathsf{SFPK}}, g_2) = e(B, Y_2) \cdot e(\mathsf{H}(m), \mathsf{Sig}^3_{\mathsf{SFPK}}).$$

Scheme 1: Our SFPK Signature Scheme

Split signing. Scheme 1 requires the signer to perform operations in $\mathbb{G}_2$ which are usually inefficient on constrained devices and influence the execution time significantly. We will now describe a technique that allows splitting the signing procedure between two parties. We will later identify them by the core and helper devices. The party holding the secret key (core) performs only operations in $\mathbb{G}_1$ and creates pre-signatures that are finalized by the second party (helper). Unforgeability of the scheme will hold against the helper device but we will require the core to perform a one-time-only pre-computation that will involve operations in $\mathbb{G}_2$. More formally.

Definition 3.1. We say that a SFPK scheme supports *split signing* if the SFPK.Sign algorithm can be divided into three steps: $\mathsf{SFPK.Sign}_1$, $\mathsf{SFPK.Sign}_2$, $\mathsf{SFPK.Sign}_3$, such that:

$\mathsf{SFPK.Sign}_1$: takes as input the security parameters 1^λ and outputs a secret state $\mathsf{st}_{\mathsf{secr}}$ and a public state $\mathsf{st}_{\mathsf{pub}}$.

$\mathsf{SFPK.Sign}_2$: takes the same inputs as SFPK.Sign and additionally $\mathsf{st}_{\mathsf{secr}}$ and outputs a pre-signature $\mathsf{pSig}_{\mathsf{SFPK}}$.

$\mathsf{SFPK.Sign}_3$: on input a pre-signature $\mathsf{pSig}_{\mathsf{SFPK}}$ and the public state $\mathsf{st}_{\mathsf{pub}}$ this algorithm outputs the final signature $\mathsf{Sig}_{\mathsf{SFPK}}$.

Additionally, we require that 1) the distribution of signatures output by $\mathsf{SFPK.Sign}_3$ is identical to the output of SFPK.Sign, 2) unforgeability holds with respect to pre-signatures even if a pair $(\mathsf{st}_{\mathsf{secr}}, \mathsf{st}_{\mathsf{pub}})$ is reused, i.e., both signing oracles in the unforgeability experiment are initialized with an output of $\mathsf{SFPK.Sign}_1$ and output pre-signatures instead of full-signatures.

We will now sketch the idea how to split the signing procedure in Scheme 1. We will use the core/helper naming convention to describe the two parties.

The only operation in $\mathbb{G}_2$ performed during signing is the computation of g_2^r. Since r is a random value, it suggests that the core can just send it to the helper and let it compute $\mathsf{Sig}^3_{\mathsf{SFPK}}$ (and even $\mathsf{Sig}^2_{\mathsf{SFPK}}$). Unfortunately, this idea fails completely because the helper would be able to extract the secret key Y_1^x from $\mathsf{Sig}^1_{\mathsf{SFPK}}$, since it can compute $\mathsf{H}(m)^r$. It is obvious that the randomness r must be kept secret and must not leak to the helper.

Our approach is now to hide r by pre-computing a value in $\mathbb{G}_2$, namely $U = g_2^u$ for $u \xleftarrow{\$} \mathbb{Z}_p^*$. The core retains u, and shares U with the helper. To sign a message, the core does not compute $\mathsf{Sig}^3_{\mathsf{SFPK}}$ but chooses $k_u \xleftarrow{\$} \mathbb{Z}_p^*$ and sends it together with $(\mathsf{Sig}^1_{\mathsf{SFPK}}, \mathsf{Sig}^2_{\mathsf{SFPK}})$ to the helper, who finalizes the signature by computing $\mathsf{Sig}^3_{\mathsf{SFPK}} = U^{k_u}$. To minimize the number of operations in $\mathbb{G}_1$ the core can use the same idea for $\mathsf{Sig}^2_{\mathsf{SFPK}}$, i.e., it can send g_1^u to the helper, which can use k_u to compute $\mathsf{Sig}^2_{\mathsf{SFPK}}$.

To show that Scheme 1 supports split signing let:

$\mathsf{SFPK.Sign}_1(1^\lambda)$: choose $k \xleftarrow{\$} \mathbb{Z}_p^*$, set $(\mathsf{st}_{\mathsf{secr}}, \mathsf{st}_{\mathsf{pub}}) = (k, (g_1^k, g_2^k))$.

$\mathsf{SFPK.Sign}_2(\mathsf{sk}_{\mathsf{SFPK}}, m, \mathsf{st}_{\mathsf{secr}})$: choose $r \xleftarrow{\$} \mathbb{Z}_p^*$ and return the pre-signature $\mathsf{pSig}_{\mathsf{SFPK}} = (Y_1^x \cdot \mathsf{H}(m)^r, r \cdot k^{-1})$.

$\mathsf{SFPK.Sign}_3(\mathsf{pSig}_{\mathsf{SFPK}}, \mathsf{st}_{\mathsf{pub}})$: parse $\mathsf{pSig}_{\mathsf{SFPK}} = (\mathsf{Sig}^1_{\mathsf{SFPK}}, w)$, $\mathsf{st}_{\mathsf{pub}} = (U_1, U_2)$ and output $(\mathsf{Sig}^1_{\mathsf{SFPK}}, U_1^w, U_2^w)$.

It is easy to see that the only difference between SFPK.Sign and the combination $(\mathsf{SFPK.Sign}_1, \mathsf{SFPK.Sign}_2, \mathsf{SFPK.Sign}_3)$ is the way $\mathsf{Sig}^2_{\mathsf{SFPK}}$ and $\mathsf{Sig}^3_{\mathsf{SFPK}}$ are computed. However, since r is chosen at random in $\mathsf{SFPK.Sign}_2$ and $U_1^w = g_1^r$ and $U_2^w = g_2^r$ are distributed identical to the output of SFPK.Sign. The main difficulty is to show that unforgeability holds in the sense as defined in Definition 3.1.

Theorem 3.2 (Unforgeability). *Scheme 1 is an unforgeable* SFPK *scheme with split signing in the random oracle model assuming the bilinear decisional Diffie-Hellman assumption.*

Proof. The proofs follows a similar strategy to the proof in [2], but with small changes due to split signing. For completeness we present the full proof of Theorem 3.2 in Appendix B.1. □

The following readily follows from [2].

Theorem 3.3 (Class-hiding). *Scheme 1 is class-hiding with key corruption in the random oracle model assuming the decisional Diffie-Hellman assumption.*

Lemma 3.4 (Canonical Representative). *A predicate defined as* $\mathsf{canon}((A, B)) := A \equiv g_1$ *can be used to identify canonical representatives in Scheme 1. Note that by defining* canon *this way the* SFPK.KeyGen *algorithm outputs keys in canonical representation.*

Third party re-randomization. A useful property that was not defined in previous work on SFPK is re-randomization of the *full* signature/public key pair. In the original work, the authors consider changing representation of the public key before the actual signature. We show that there exists an algorithm $(\mathsf{pk}'_{\mathsf{SFPK}}, \mathsf{Sig}'_{\mathsf{SFPK}}) \leftarrow \mathsf{SFPK.ReRand}(\mathsf{pk}_{\mathsf{SFPK}}, m, \mathsf{Sig}_{\mathsf{SFPK}}, r)$ for which we have $\mathsf{pk}'_{\mathsf{SFPK}} \leftarrow \mathsf{SFPK.ChgPK}(\mathsf{pk}_{\mathsf{SFPK}}, r)$ and $\mathsf{SFPK.Verify}(\mathsf{pk}'_{\mathsf{SFPK}}, m, \mathsf{Sig}'_{\mathsf{SFPK}}) = 1$ where for the original signature $\mathsf{SFPK.Verify}(\mathsf{pk}_{\mathsf{SFPK}}, m, \mathsf{Sig}_{\mathsf{SFPK}}) = 1$. We can define this algorithm as part of Scheme 1 as follows:

$\mathsf{SFPK.ReRand}(\mathsf{pk}_{\mathsf{SFPK}}, m, \mathsf{Sig}_{\mathsf{SFPK}}, r)$: parse $\mathsf{Sig}_{\mathsf{SFPK}} = (\mathsf{Sig}^1_{\mathsf{SFPK}}, \mathsf{Sig}^2_{\mathsf{SFPK}}, \mathsf{Sig}^3_{\mathsf{SFPK}})$, choose random $k \xleftarrow{\$} \mathbb{Z}_p^*$, compute $\mathsf{pk}'_{\mathsf{SFPK}} \leftarrow \mathsf{SFPK.ChgPK}(\mathsf{pk}_{\mathsf{SFPK}}, r)$ and set $\mathsf{Sig}'_{\mathsf{SFPK}} = ((\mathsf{Sig}^1_{\mathsf{SFPK}})^r \cdot \mathsf{H}(m)^k, (\mathsf{Sig}^2_{\mathsf{SFPK}})^r \cdot g_1^k, (\mathsf{Sig}^3_{\mathsf{SFPK}})^r \cdot g_2^k)$.

3.2 Tag-Based Equivalence Class Signatures

Now, we introduce a variant of SPS-EQ or more precisely equivalence class signatures (as they are not strictly structure-preserving anymore) that in addition to the message M being a representative of class $[M]$ support an auxiliary tag $\tau \in \{0,1\}^*$. Therefore, we adapt the security model from SPS-EQ as follows. The task of the adversary is to forge a signature for a message (M^*, τ^*) where the adversary did not query a signature for the class $[M^*]$ and τ^* combination (cf. Appendix A.4). Moreover, for the adaption notion which guarantees that signatures from ChgRep and Sign are identically distributed, we only require it to hold with respect to identical auxiliary tags τ. Our construction is a modification of the SPS-EQ scheme from [44] (denoted FHS15 henceforth) which is proven to be EUF-CMA secure in the generic group model and provides perfect adaption even under malicious keys. We do not provide an abstract definition as the only changes to the SPS-EQ interface are the additional input τ to the Sign and Verify algorithms. Our construction of a tag-based equivalence class signature scheme (TBEQ) is provided in Scheme 2 and it basically extends the FHS15 scheme by a fourth signature element $V_2 = H(\tau)^{\frac{1}{y}}$ where $H : \{0,1\}^* \to \mathbb{G}_2$ is modeled as a random oracle and y is the signing randomness. Note that V_2 can be considered as a BLS signature [16] with the signing randomness $1/y$ acting as a one-time signing key.
We will now show the unforgeability and perfect adaption of the TBEQ in Scheme 2.

TBEQ.Setup(1^λ): generate $\mathsf{BG} \xleftarrow{\$} \mathsf{BGGen}(\lambda)$, $H : \{0,1\}^* \to \mathbb{G}_2$ and return $\mathsf{params} = (\mathsf{BG}, H)$.

TBEQ.KeyGen(params, ℓ): choose $\vec{x} \xleftarrow{\$} (\mathbb{Z}_p^*)^\ell$ and set $\mathsf{sk} = \vec{x}$ and $\mathsf{pk} = g_2^{\vec{x}} = (g_2^{x_1}, \ldots, g_2^{x_\ell})$.

TBEQ.Sign(sk, M, τ): parse $\mathsf{sk} = \vec{x}$, $M \in (\mathbb{G}_1^*)^\ell$, $\tau \in \{0,1\}^*$ and choose $y \xleftarrow{\$} \mathbb{Z}_p$. Compute

$$Z_1 = \left(\prod_{i=1}^{\ell} M_i^{x_i}\right)^y, \; Y_1 = g_1^{\frac{1}{y}}, \; Y_2 = g_2^{\frac{1}{y}} \text{ and } V_2 = H(\tau)^{\frac{1}{y}}.$$

Return $\sigma = (Z_1, Y_1, Y_2, V_2)$.

TBEQ.ChgRep(M, σ, μ, pk): Choose $\psi \xleftarrow{\$} \mathbb{Z}_p^*$ and return (M^μ, σ') with $\sigma' = (Z_1^{\psi\mu}, Y_1^{\frac{1}{\psi}}, Y_2^{\frac{1}{\psi}}, V_2^{\frac{1}{\psi}})$.

TBEQ.Verify(pk, M, τ, σ): parse $\mathsf{pk} = (\mathsf{pk}_1 = g_2^{x_1}, \ldots, \mathsf{pk}_\ell = g_2^{x_\ell})$, $M \in (\mathbb{G}_1^*)^\ell$, $\tau \in \{0,1\}^*$ and $\sigma = (Z_1, Y_1, Y_2, V_2)$. Return 1 if the following checks hold and 0 otherwise:

$$\prod_{i=1}^{\ell} e(M_i, \mathsf{pk}_i) = e(Z_1, Y_2) \;\wedge$$
$$e(Y_1, g_2) = e(g_1, Y_2) \;\wedge\; e(g_1, V_2) = e(Y_1, H(\tau))$$

Scheme 2: Our TBEQ Signature Scheme

Theorem 3.5. *The* TBEQ *in Scheme 2 is EUF-CMA secure and provides perfect adaption (under malicious keys) assuming that H is a random oracle.*

We argue unforgeability in the generic bilinear group model (following the proof of the FHS15 SPS-EQ in [45]) for a version of our TBEQ without random oracles and a polynomially bounded tag-space. Then, we will argue our modification in the random oracle model with an unbounded tag space and constant size public keys. The idea for a polynomially bounded tag space $\mathcal{T} = \{\tau_1, \ldots, \tau_k\}$ for a $k \in \mathsf{poly}(\lambda)$ is to include additional uniformly random elements $(h_i \in \mathbb{G}_2)_{i \in [k]}$ into pk and use the corresponding value h_i when signing for tag τ_i instead of the hash evaluation $H(\tau_i)$.

Lemma 3.6. *The* TBEQ *in Scheme 2 with the above modifications is EUF-CMA secure in the Type-3 generic bilinear group model.*

We provide this proof in Appendix B.2.

Lemma 3.7. *The* TBEQ *in Scheme 2 is EUF-CMA secure for an unbounded tag-space when modeling H as a random oracle.*

Proof. Up to collisions in the random oracle, which happen with negligible probability, the TBEQ in Scheme 2 and in particular the security analysis is identical to the proof of Lemma 3.6, but without the restriction of the tag space being polynomial in size. □

Lemma 3.8. *The* TBEQ *in Scheme 2 provides perfect adaption (under malicious keys).*

We provide this proof in Appendix B.3.

What we require for our further constructions is a derandomized version of the TBEQ scheme. Subsequently, we formulate as Lemma 3.9 (cf. [17]) a frequently used technique (see e.g., [9, 52]) to derandomize any signature scheme, which in particular also holds for TBEQ. Thus, we omit the proof.

Lemma 3.9. *Let* $\Sigma = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{ChgRep}, \mathsf{Verify})$ *be an EUF-CMA secure* TBEQ *scheme and* $F : \mathcal{K} \times \mathcal{M}_{\mathsf{TBEQ}} \to \mathcal{R}_{\mathsf{TBEQ}}$ *be a secure PRF, then* $\Sigma' = (\Sigma.\mathsf{Setup}, \mathsf{KeyGen}', \mathsf{Sign}', \Sigma.\mathsf{ChgRep}, \Sigma.\mathsf{Verify})$ *is also EUF-CMA secure, where:*

KeyGen′(BG, ℓ): *Run* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \Sigma.\mathsf{KeyGen}(\mathsf{BG}, \ell)$*, choose* $k \xleftarrow{\$} \mathcal{K}$ *and return* $((\mathsf{sk}, k), \mathsf{pk})$.

Sign′(sk, M, τ): *Compute* $r := F(k, M)$ *and return* $\Sigma.\mathsf{Sign}(\mathsf{sk}, M, \tau; r)$.

We denote the derandomized TBEQ by TBEQ_d. Note that in Scheme 2 this means that in Sign we have $y \leftarrow F(k, M)$.

3.3 Aggregatable Attribute-Based EQs

We now introduce another variant of equivalence class signatures called aggregatable attribute-based equivalence class (AAEQ) signatures, that will represent one core building block for our CHAC system. In such a scheme there is a main key pair, which is akin to identity-based signatures [71]. The main secret key can issue signing keys for attributes (Attr), e.g., Attr ="age". When signing a message M (a representative of a class $[M]$) with respect to such an attribute signing key, signing additionally takes an attribute value v_{Attr}, e.g., v_{Attr} ="21". The scheme is required to be aggregatable in a sense that signatures under different attribute signing keys for the same *representative* M of a class can be aggregated into a compact signature. Like in SPS-EQ, the signatures are with respect to classes and there is a ChgRep algorithm to publicly change representatives (i.e., adapt). For the sake of simplicity, below we assume that the set of attributes represents the integers $[t]$ with domain $\{0,1\}^*$ for each attribute.

Definition 3.10 (Aggregatable Attribute-Based EQs). An aggregatable attribute-based equivalence class (AAEQ) signature scheme consists of the following PPT algorithms:

Setup($1^\lambda, t, \ell$): on input security parameter 1^λ, the number of attributes t (distinct attribute names) and length parameter ℓ this algorithm outputs main key pair (msk, mpk).

AKGen(msk, Attr): on input a main secret key msk and an attribute Attr, outputs an attribute secret key $\mathsf{sk}_{\mathsf{Attr}}$.

Sign($\mathsf{sk}_{\mathsf{Attr}}, v_{\mathsf{Attr}}, M$): on input an attribute secret key $\mathsf{sk}_{\mathsf{Attr}}$, an attribute value v_{Attr} and a representative M, this algorithm outputs a signature σ.

ChgRep(M, σ, μ, mpk): on input a representative M, a signature σ, a scalar μ and a main public key mpk, this algorithm outputs an updated signature σ' for representative M^μ.

Agg(mpk, $\{\sigma_i\}$): on input a main public key mpk and a set of valid signatures $\{\sigma_i\}$, outputs an aggregated signature σ'.

Verify(mpk, $\{\mathsf{Attr}_i\}, \sigma', M$): on input a public key mpk, a set of attributes $\{(\mathsf{Attr}_i, v_{\mathsf{Attr}_i})\}$, an aggregated signature σ' and a representative M, outputs either accept(1) or reject(0).

We require an AAEQ to be correct, unforgeable and to provide perfect adaption. We present the formal definitions in Appendix A.5.

Intuition of our construction. We now present a construction with $O(\lambda)$ sized mpk and msk as Scheme 3 which is based upon the TBEQ in Scheme 2 using the de-randomization (TBEQ_d). The idea is simple and uses parallel instances of the derandomized TBEQ_d scheme, where every pk represents a different attribute Attr (for simplicity just integers in the set $[t]$, but this can easily be changed to arbitrary strings, e.g., Attr ="age"). Now the basic idea is to use the attribute value v_{Attr} as the tag in the TBEQ scheme.

The intuition is that signatures for multiple different attributes and the same representative M of class $[M]$ share the same randomness $y = F(k, M)$ and thus from the set of w signatures $\{(Z_{1,i}, Y_{1,i}, Y_{2,i}, V_{2,i})\}_{i \in [w]}$ aggregation can easily be done by aggregating the $Z_{1,i}$ components of all single signatures as well as the $V_{2,i}$ components and use the Y_1, Y_2 values of one of the signatures (note that all with respect to the same mpk and same representative M use the same randomness y and are thus identical). Aggregate verification is the verification of the TBEQ scheme using the componentwise aggregation of the attribute public keys (see Scheme 3 for details). Finally, the change representative algorithm is identical to the algorithm of the underlying TBEQ. Note that for the simplicity of presentation we assume that ChgRep and Agg only take *valid* signatures as input (this can easily be handled by adding verification of all input signatures to the respective algorithms).

<u>AAEQ.Setup$(1^\lambda, t, \ell)$:</u> generate BG $\xleftarrow{\$}$ BGGen(λ), choose $H : \{0,1\}^* \rightarrow \mathbb{G}_2$ and set params $= (\mathsf{BG}, H)$. Choose PRF key $k \xleftarrow{\$} \mathcal{K}$ and for $i \in [t]$

- choose $\vec{x}_i \xleftarrow{\$} (\mathbb{Z}_p^*)^\ell$, set $\mathsf{pk}_{\mathsf{Attr}_i} = (g_2^{\vec{x}_i})$ and set $\mathsf{sk}_{\mathsf{Attr}_i} = (\mathsf{pk}_{\mathsf{Attr}_i}, \vec{x}_i, \mathsf{k})$.

Set $\mathsf{msk} = (\mathsf{sk}_{\mathsf{Attr}_1}, \ldots, \mathsf{sk}_{\mathsf{Attr}_t})$ and $\mathsf{mpk} = (\mathsf{pk}_{\mathsf{Attr}_1}, \ldots, \mathsf{pk}_{\mathsf{Attr}_t})$ and return $(\mathsf{msk}, \mathsf{mpk})$.

<u>AAEQ.AKGen(msk, Attr):</u> parse $\mathsf{msk} = (\mathsf{sk}_{\mathsf{Attr}_1}, \ldots, \mathsf{sk}_{\mathsf{Attr}_t})$ and Attr $\in [t]$ and return msk[Attr].

<u>AAEQ.Sign$(\mathsf{sk}_{\mathsf{Attr}}, v_{\mathsf{Attr}}, M)$:</u> parse $\mathsf{sk}_{\mathsf{Attr}} = (\mathsf{pk}_{\mathsf{Attr}}, \vec{x}, k)$, $v_{\mathsf{Attr}} \in \{0,1\}^*$, $M \in (\mathbb{G}_1^*)^\ell$, compute $y \leftarrow F(k, M)$ and with $H_{\mathsf{Attr}}(\cdot) := H(\mathsf{pk}_{\mathsf{Attr}} \| \cdot)$ compute

$$Z_1 = \left(\prod_{i=1}^{\ell} M_i^{x_i}\right)^y, \; Y_1 = g_1^{\frac{1}{y}}, Y_2 = g_2^{\frac{1}{y}}, V_2 = H_{\mathsf{Attr}}(v_{\mathsf{Attr}})^{\frac{1}{y}}.$$

Return $\sigma = (Z_1, Y_1, Y_2, V_2) \in (\mathbb{G}_1^*)^2 \times (\mathbb{G}_2^*)^2$.

<u>AAEQ.ChgRep$(M, \sigma, \mu, \mathsf{mpk})$:</u> given $M \in (\mathbb{G}_1^*)^\ell$, a valid signature σ, $\mu \in \mathbb{Z}_p^*$ and mpk, choose $\psi \xleftarrow{\$} \mathbb{Z}_p^*$ and return (M^μ, σ') with $\sigma' = (Z_1^{\psi\mu}, Y_1^{\frac{1}{\psi}}, Y_2^{\frac{1}{\psi}}, V_2^{\frac{1}{\psi}})$.

<u>AAEQ.Agg$(\mathsf{mpk}, \{\sigma_i\})$:</u> given mpk and set of valid signatures $\{\sigma_i\}$ of size k parse it as $\sigma_i = (Z_{1,i}, Y_{1,i}, Y_{2,i}, V_{2,i})$ and return $\perp$ if $Y_{1,i} \neq Y_{1,j}$ or $Y_{2,i} \neq Y_{2,j}$ for $i \neq j$, $i, j \in [k]$ and otherwise return $(\prod_{i=1}^{k} Z_{1,i}, Y_{1,1}, Y_{2,1}, \prod_{i=1}^{k} V_{2,i})$.

<u>AAEQ.Verify$(\mathsf{mpk}, \{\mathsf{Attr}\}, \sigma', M)$:</u> parse $\mathsf{mpk} = (\mathsf{pk}_{\mathsf{Attr}_1}, \ldots, \mathsf{pk}_{\mathsf{Attr}_t})$, $\{\mathsf{Attr}\} = ((\mathsf{Attr}_i, v_{\mathsf{Attr}_i}))_{i \in [t]} \in ([t] \times \{0,1\}^*)^k$, $\sigma' = (Z_1, Y_1, Y_2, V_2)$ and $M \in (\mathbb{G}_1^*)^\ell$. Return 1 if the following checks hold and 0 otherwise:

$$\prod_{i=1}^{\ell} e(M_i, \prod_{j=1}^{k} \mathsf{pk}_{\mathsf{Attr}_{j,i}}) = e(Z_1, Y_2) \wedge e(Y_1, g_2) = e(g_1, Y_2) \wedge$$

$$e(Y_1, \prod_{j=1}^{k} H_{\mathsf{Attr}_j}(v_{\mathsf{Attr}_j})) = e(g_1, V_2)$$

Scheme 3: Our AAEQ **Signature Scheme**

Now, we prove the security of our AAEQ scheme in Scheme 3.

THEOREM 3.11. *The* AAEQ *scheme in Scheme 3 is EUF-CMA and provides perfect adaption assuming that H is a random oracle.*

We again prove the above theorem using a sequence of lemmas.

LEMMA 3.12. *The* AAEQ *scheme in Scheme 3 with bounded attribute-space is EUF-CMA secure in the generic bilinear group model for Type-3 bilinear groups.*

The proof is given in Appendix B.4.

LEMMA 3.13. *The* AAEQ *in Scheme 3 is EUF-CMA secure for an unbounded attribute-space when modeling H as a random oracle.*

PROOF. Up to collisions in the random oracle, which happen with negligible probability, the AAEQ in Scheme 3 and in particular the analysis is identical to the proof of Lemma 3.12, but without the restriction of the tag space being polynomial in size. □

LEMMA 3.14. *The* AAEQ *scheme in Scheme 3 provides perfect adaption if the* $\mathsf{TBEQ_d}$ *Scheme 2 provides perfect adaption.*

PROOF. This straightforwardly follows from the perfect adaption notion of the underlying $\mathsf{TBEQ_d}$ scheme. □

4 CORE/HELPER CREDENTIALS

We recall that in ACs usually a personal computer or smartphone is used to store and show the credential and it is assumed that the user's device is not limited in any way, i.e., computational or communication-wise. A core/helper anonymous credential (CHAC) system considers a different and more realistic scenario. We consider two devices, a core device with limited capabilities (i.e., small memory and computational power) and a helper device that is more powerful and the only gateway of the core device to the outside world, e.g., the Internet. The core device creates and stores the secret key required to show credentials. However, since it is limited it only creates so-called partial show tokens. The helper device stores the credentials and finalizes the show token. The key idea here is that the core device is responsible for protecting credentials (i.e., the key to use them) and the helper device is responsible for protecting the privacy of the showing procedure. In CHACs we will only consider single round communications and therefore the semantic will consist only of algorithms and not protocols as it is the case in standard anonymous credentials.

4.1 Syntax and Security Model

Before defining the syntax of a CHAC system, we assume that there exists a compressing and collision-resistant function AIDGen (Attr, nonce) that on input a non-empty attribute set Attr and random nonce $\in \{0,1\}^\lambda$, outputs an attribute identifier aid $\in \{0,1\}^\lambda$. We will assume that the attribute set Attr contains pairs of a name and value, e.g. a valid element is `('Age:', '18')`.

*Definition 4.1 (*CHAC*).* A core/helper anonymous credential (CHAC) system consists of the following PPT algorithms:

Setup$_{\mathsf{CHAC}}(1^\lambda)$: on input security parameter 1^λ, this algorithm outputs a common reference string ρ, which is an implicit input to the below algorithms. Some constructions might not require such a string and work without a trusted setup.

IKGen(1^λ): on input security parameter 1^λ, this algorithm outputs the issuer's key pair (isk, ipk).

CKGen(1^λ): on input security parameter 1^λ, this algorithm outputs the core device secret key ssk.

$\mathsf{CObtain}(\mathsf{aid}, \mathsf{ipk}, \mathsf{ssk})$: on input attribute identifier aid, issuer's public key ipk and secret key ssk, executed by the core device outputs a partial credential request apreq.

$\mathsf{HObtain}(\mathsf{Attr}, \mathsf{nonce}, \mathsf{ipk}, \mathsf{apreq})$: on input non-empty attribute set Attr, a random nonce $\in \{0, 1\}^\lambda$, issuer's public key ipk and partial credential request apreq, this algorithm executed by the helper outputs a credential request areq.

$\mathsf{Issue}(\mathsf{Attr}, \mathsf{nonce}, \mathsf{areq}, \mathsf{isk})$: on input non-empty attribute set Attr, a random nonce $\in \{0, 1\}^\lambda$, credential request areq and issuer's secret key isk, this algorithm outputs $\perp$ on failure and otherwise a credential cred and a device identifier did.

$\mathsf{CShow}(\mathsf{aid}, \mathsf{ipk}, \mathsf{ssk})$: on input attribute identifier aid, issuer's public key ipk and secret key ssk, this algorithm executed by the core device outputs a partial show token apsig.

$\mathsf{HShow}(\mathsf{Attr}, \mathsf{nonce}, \mathsf{cred}, \mathsf{ipk}, \mathsf{apsig})$: on input non-empty attribute set Attr, a random nonce $\in \{0, 1\}^\lambda$, credential cred, issuer's public key and partial show token apsig, this algorithm executed by the helper outputs a full show token asig.

$\mathsf{Verify}(\mathsf{Attr}, \mathsf{nonce}, \mathsf{asig}, \mathsf{ipk})$: on input non-empty attribute set Attr, a nonce $\in \{0, 1\}^\lambda$, full show token asig and issuer's public key, this algorithm outputs either accept(1) or reject(0).

We say that a core/helper anonymous credential system is *secure* if it is correct, unforgeable, dependable, anonymous and compact.

Correctness. As one would expect, a showing of a credential with respect to a non-empty set Attr_D of attributes always verifies if the credential was issued honestly for some attribute set Attr_A with $\mathsf{Attr}_D \subseteq \mathsf{Attr}_A$.

Unforgeability. Showing of attributes for which one does not possess credentials should not be possible. Even a malicious coalition should be unable to combine their credentials and show a set of attributes that no single member has.

Dependability. An adversary that takes control over the helper device should be unable to show an honestly generated credential in a given session without interaction with the core device, i.e. this involves the case that credentials stored on the helper device leak.

Anonymity. A coalition of a malicious verifier and issuer should not be able to identify the core/helper devices, except that they possess a valid credential for the shown attributes. Furthermore, different showings of the same credential should be unlinkable.

Compactness. The size of the full show token asig should not depend on the number of attributes.

Formal definitions of those properties are given in Appendix A.6.

4.2 Generic Construction

We will now present our generic construction of a CHAC system for up to t attributes i.e., the upper bound on the number of different attributes an issuer can issue. The two main building blocks are a SFPK scheme with public key size ℓ and split signing, and an AAEQ scheme with message size ℓ. We assume that the space of SFPK public keys and AAEQ messages are compatible (the same). We also assume that the SFPK key generation algorithm outputs public keys in canonical form.

Our construction uses the idea of self-blindable certificates similar to [63]. The core device generates a long-term SFPK key pair that is used for all credentials. This key pair is used as a standard signing key and the core device does not use the randomization properties of the SFPK public key. However, this key is "certified" by the issuer using the AAEQ scheme. Since it is attribute-based, the issuer can easily create multiple signatures on the core device's public key depending on the possessed attributes. A credential is then formed by appending all signatures, i.e., its size depends on the number of attributes. To show an attribute the core device uses the SFPK signing procedure to sign an attribute identifier aid send by the helper device and which corresponds to the disclosed attributes Attr and a nonce (from the verifier). Once the helper device receives the SFPK signature from the core device it finalizes (we use split signing here) and randomizes it. We will use n to denote the number of attributes that were issued to a user and by $k \leq n$ the number of attributes that are selectively disclosed within a show token. Additionally, it aggregates all AAEQ signatures that correspond to the shown attributes (i.e., the k that should be selectively disclosed) and uses the same random coins to randomize it. Note that thanks to aggregation the show tokens size is independent of the number of shown attributes. The final show token is a random SFPK public key, the corresponding SFPK signature under $\mathsf{aid} = \mathsf{AIDGen}(\mathsf{Attr}, \mathsf{nonce})$ and an aggregated AAEQ signature for the public key. More details are given in Scheme 4.

We now show that Scheme 4 can be efficiently instantiated in the random oracle model using an SFPK with split signing and an AAEQ scheme (cf. Section 3).

Theorem 4.2 (Unforgeability). *Scheme 4 is unforgeable assuming the used* SFPK *with split signing is unforgeable, the used* AAEQ *is unforgeable and* AIDGen *is collision-resistant.*

Theorem 4.3 (Anonymity). *Scheme 4 is anonymous if the used* AAEQ *are adaptable and the* SFPK *signatures are class-hiding.*

Theorem 4.4 (Dependability). *Scheme 4 is dependable if* SFPK *with split signing is unforgeable and* AIDGen *is collision-resistant.*

For completeness the proofs for unforgeability, anonymity and dependability are given respectively in Appendix C.1, C.2 and C.3.

Remark. For our concrete instantiation in the next section, we require that for every user SFPK public key all requested attributes are queried once and at the same time. While this is a proof artifact to simplify the GGM proof, we 1) do not expect this to be a problem for most use-cases and 2) conjecture that even if ignored this implies no issues with the security of the CHAC construction.

5 CHAC EVALUATION

In this section we evaluate a concrete instantiation of our CHAC system based on the building blocks from Section 3. Moreover, discuss techniques used to optimize the smart card implementation and helper device side of the CHAC system.

5.1 Setup

To evaluate our CHAC system we prepared a prototype implementation. We used a Multos smart card [60] as the core device and implement the helper device on a smartphone with a Snapdragon

$\underline{\mathsf{Setup}_{\mathsf{CHAC}}(1^\lambda):}$ return $\rho \leftarrow \mathsf{SFPK.CRSGen}(1^\lambda)$.

$\underline{\mathsf{IKGen}(1^\lambda):}$ return $(\mathsf{isk}, \mathsf{ipk}) \leftarrow \mathsf{AAEQ.Setup}(1^\lambda, t, \ell)$.

$\underline{\mathsf{CKGen}(1^\lambda):}$ choose $(\mathsf{sk}_{\mathsf{SFPK}}, \mathsf{pk}_{\mathsf{SFPK}}) \leftarrow \mathsf{SFPK.KeyGen}(1^\lambda)$ and compute $(\mathsf{st}_{\mathsf{secr}}, \mathsf{st}_{\mathsf{pub}}) \leftarrow \mathsf{SFPK.Sign}_1(1^\lambda)$. Return $\mathsf{ssk} = (\mathsf{sk}_{\mathsf{SFPK}}, \mathsf{pk}_{\mathsf{SFPK}}, \mathsf{st}_{\mathsf{secr}}, \mathsf{st}_{\mathsf{pub}})$.

$\underline{\mathsf{CObtain}(\mathsf{aid}, \mathsf{ipk}, \mathsf{ssk}):}$ parse $\mathsf{ssk} = (\mathsf{sk}_{\mathsf{SFPK}}, \mathsf{pk}_{\mathsf{SFPK}}, \mathsf{st}_{\mathsf{secr}}, \mathsf{st}_{\mathsf{pub}})$, compute $\mathsf{pSig}_{\mathsf{SFPK}} \leftarrow \mathsf{SFPK.Sign}_2(\mathsf{sk}_{\mathsf{SFPK}}, \mathsf{aid}, \mathsf{st}_{\mathsf{secr}})$ and return $\mathsf{apreq} = (\mathsf{pk}_{\mathsf{SFPK}}, \mathsf{st}_{\mathsf{pub}}, \mathsf{pSig}_{\mathsf{SFPK}})$.

$\underline{\mathsf{HObtain}(\mathsf{Attr}, \mathsf{nonce}, \mathsf{ipk}, \mathsf{apreq}):}$ parse $\mathsf{apreq} = (\mathsf{pk}_{\mathsf{SFPK}}, \mathsf{st}_{\mathsf{pub}}, \mathsf{pSig}_{\mathsf{SFPK}})$, compute $\mathsf{Sig}_{\mathsf{SFPK}} \leftarrow \mathsf{SFPK.Sign}_3(\mathsf{pSig}_{\mathsf{SFPK}}, \mathsf{st}_{\mathsf{pub}})$ and return $\mathsf{areq} = (\mathsf{pk}_{\mathsf{SFPK}}, \mathsf{Sig}_{\mathsf{SFPK}})$.

$\underline{\mathsf{Issue}(\mathsf{Attr}, \mathsf{nonce}, \mathsf{areq}, \mathsf{isk}):}$ parse $\mathsf{Attr} = \{(\mathsf{Attr}_1, v_{\mathsf{Attr}_1}), \ldots, (\mathsf{Attr}_n, v_{\mathsf{Attr}_n})\}$, $\mathsf{areq} = (\mathsf{pk}_{\mathsf{SFPK}}, \mathsf{Sig}_{\mathsf{SFPK}})$ and $\mathsf{isk} = \mathsf{msk}$.

- Compute identifier $\mathsf{aid} = \mathsf{AIDGen}(\mathsf{Attr}, \mathsf{nonce})$ and output $\perp$ if $\mathsf{SFPK.Verify}(\mathsf{pk}_{\mathsf{SFPK}}, \mathsf{aid}, \mathsf{Sig}_{\mathsf{SFPK}}) = 0$ or $\mathsf{canon}(\mathsf{pk}_{\mathsf{SFPK}}) \neq 1$.
- For all indices $i \in \{1, \ldots, n\}$ recompute the AAEQ keys $\mathsf{sk}_{\mathsf{Attr}_i} \leftarrow \mathsf{AAEQ.AKGen}(\mathsf{msk}, \mathsf{Attr}_i)$ and compute signatures $\sigma_{\mathsf{Attr}_i} \leftarrow \mathsf{AAEQ.Sign}(\mathsf{sk}_{\mathsf{Attr}_i}, v_{\mathsf{Attr}_i}, \mathsf{pk}_{\mathsf{SFPK}})$.
- Output $\mathsf{cred} = (\sigma_{\mathsf{Attr}_1}, \ldots, \sigma_{\mathsf{Attr}_n})$ and $\mathsf{did} = \mathsf{pk}_{\mathsf{SFPK}}$.

$\underline{\mathsf{CShow}(\mathsf{aid}, \mathsf{ipk}, \mathsf{ssk}):}$ execute $\mathsf{apsig} \leftarrow \mathsf{CObtain}(\mathsf{aid}, \mathsf{ipk}, \mathsf{ssk})$.

$\underline{\mathsf{HShow}(\mathsf{Attr}, \mathsf{nonce}, \mathsf{cred}, \mathsf{ipk}, \mathsf{apsig}):}$ parse $\mathsf{Attr} = \{(\mathsf{Attr}_1, v_{\mathsf{Attr}_1}), \ldots, (\mathsf{Attr}_k, v_{\mathsf{Attr}_k})\}$ $\mathsf{apsig} = (\mathsf{pk}_{\mathsf{SFPK}}, \mathsf{pSig}_{\mathsf{SFPK}}, \mathsf{st}_{\mathsf{pub}})$.

- Compute identifier $\mathsf{aid} = \mathsf{AIDGen}(\mathsf{Attr}, \mathsf{nonce})$ and finalize signature $\mathsf{Sig}_{\mathsf{SFPK}} \leftarrow \mathsf{SFPK.Sign}_3(\mathsf{pSig}_{\mathsf{SFPK}}, \mathsf{st}_{\mathsf{pub}})$.
- Set $\mathsf{Attr}_\sigma = \{\sigma_{\mathsf{Attr}_1}, \ldots, \sigma_{\mathsf{Attr}_k}\}$ and aggregate the AAEQ signature $\sigma_{\mathsf{Attr}} \leftarrow \mathsf{AAEQ.Agg}(\mathsf{ipk}, \mathsf{Attr}_\sigma)$.
- Compute $(\mathsf{pk}'_{\mathsf{SFPK}}, \mathsf{Sig}'_{\mathsf{SFPK}}) \leftarrow \mathsf{SFPK.ReRand}(\mathsf{pk}_{\mathsf{SFPK}}, \mathsf{aid}, \mathsf{Sig}_{\mathsf{SFPK}}, r)$ using blinding $r \xleftarrow{\$} \mathsf{coin}_{\mathsf{SFPK}}$.
- Change the representation of the signature $\sigma'_{\mathsf{Attr}} \leftarrow \mathsf{AAEQ.ChgRep}(\mathsf{pk}_{\mathsf{SFPK}}, \sigma_{\mathsf{Attr}}, r, \mathsf{ipk})$.
- Return the show token $\mathsf{asig} = (\mathsf{pk}'_{\mathsf{SFPK}}, \mathsf{Sig}'_{\mathsf{SFPK}}, \sigma'_{\mathsf{Attr}})$.

$\underline{\mathsf{Verify}(\mathsf{Attr}, \mathsf{nonce}, \mathsf{asig}, \mathsf{ipk}):}$ Parse $\mathsf{asig} = (\mathsf{pk}'_{\mathsf{SFPK}}, \mathsf{Sig}'_{\mathsf{SFPK}}, \sigma'_{\mathsf{Attr}})$ and compute $\mathsf{aid} = \mathsf{AIDGen}(\mathsf{Attr}, \mathsf{nonce})$. Return 0 if $\mathsf{SFPK.Verify}(\mathsf{pk}'_{\mathsf{SFPK}}, \mathsf{aid}, \mathsf{Sig}'_{\mathsf{SFPK}}) = 0$. Otherwise return $\mathsf{AAEQ.Verify}(\mathsf{ipk}, \{(\mathsf{Attr}_i, v_{\mathsf{Attr}_i})\}, \sigma'_{\mathsf{Attr}}, \mathsf{pk}'_{\mathsf{SFPK}})$.

Scheme 4: Our Generic Construction of CHAC

710 processor and 6GB RAM running Android 10.0. To make the evaluation more comprehensive, we executed the same helper device code on a laptop with Intel i7-7660U CPU @ 2.50 GHz with 16GB RAM running Windows 10.

We instantiate the bilinear groups using BN-256 curves [4] where the group $\mathbb{G}_1$ is a standard curve defined over $\mathbb{F}_p$, $\mathbb{G}_2$ is a curve defined over the extension field $\mathbb{F}_{p^2}$ and the target group is $\mathbb{F}_{p^{12}}$.

5.2 Implementing SFPK on a Smart Card

On a high level, to implement the core device part of the construction in Section 4.2 we have to implement the SFPK key generation (SFPK.KeyGen) and signing algorithms ($\mathsf{SFPK.Sign}_1$ and $\mathsf{SFPK.Sign}_2$). They involve the following elliptic curve operations:

SFPK.KeyGen: standard elliptic curve key generation,
$\mathsf{SFPK.Sign}_1$: point multiplication in $\mathbb{G}_1$ and $\mathbb{G}_2$,
$\mathsf{SFPK.Sign}_2$: point multiplication, addition, hashing in $\mathbb{G}_1$.

Below we describe three principles and explain in detail how we implemented the above algorithms on-card. What is more important, the described principles explain the design choices we made in the construction of our CHAC system.

Standardized operations. Multi-app smart cards usually provide a high-level programming API with standardized cryptographic algorithms and some basic operations like memory copying. We decided on Multos smart cards because they provide API access to modular arithmetic, which is not the case for the popular Java Card technology-based cards [64]. The main limitation of smart cards is that algorithms implemented directly are strongly inefficient in comparison to the ones provided by the API, e.g., Bichsel et. al. [10] used API based exponentiation (via the RSA algorithm) and the equation $(a + b)^2 = a^2 + 2ab + b^2$ to implement multiplication.

The Gemalto Multos card we used for our evaluation supports elliptic curves, but it is limited to standard curves over $\mathbb{F}_p$. There is also no support for low-level operations like point addition and multiplication. Instead, the API provides access to an elliptic curve Diffie-Hellman (ECDH) algorithm that outputs only the x-coordinate of the resulting point. Implementing point addition using the API provided modular arithmetic is sufficiently efficient.

To implement $\mathsf{SFPK.Sign}_1$ and $\mathsf{SFPK.Sign}_2$ we do not need an actual point multiplication algorithm because the scalar in both cases is random and chosen by the core device. Therefore, we can leverage the API provided elliptic curve key generation algorithm that outputs the full representation of the public key. What is more, the parameters of the curve can be easily changed and therefore we can use an arbitrary group generator that allows us to compute, e.g., $\mathsf{H}(m)^r$ by replacing the group generator by $\mathsf{H}(m)$.

It remains to discuss how one can implement operations in $\mathbb{G}_2$, since elliptic curves over an extension field $\mathbb{F}_{p^2}$ are not supported. In this case there are no API level algorithms that could be used to make a custom implementation faster. This is the main reason why we divide the SFPK signing process and included a pre-computation step $\mathsf{SFPK.Sign}_1$. Since the generation of the core's device secret key is a one-time operation and can take more time than the online signing process. Thus, point multiplication for curves over $\mathbb{F}_{p^2}$ can be implemented using the API provided modular arithmetic.

Reusable Code. Smart cards are not only constrained in terms of computation power but also in terms of memory. Usually the card provided around 100 KB for applications which consist of compiled code and defined data structured (e.g., secret keys). We took this into account while designing our construction by limiting the operations of the core device. This is also the main reason why CShow executes CObtain and on a high level, both algorithms are just $\mathsf{SFPK.Sign}_2$. What is more, this is also the reason why the core device performs operations that are independent, in some sense, of the attributes shown/obtained which allowed us to store the credentials on the helper device.

Helper device characteristics. In CHAC we consider the helper device somewhat trusted, i.e., it should be unable to use credentials without the core device but otherwise, it is considered trusted (i.e., w.r.t. privacy). We abuse this in our implementation. The first idea we introduce is how to hash the aid value to a point in $\mathbb{G}_1$. Usually, one would use techniques like Icart's function [51] to do this, but since we put some trust in the helper we can use a simpler algorithm. The idea is to limit the aid space to only values for which computing SHA-256 give a valid x-coordinate in $\mathbb{G}_1$. We also assume that the helper provides a valid y-coordinate. This approach can be easily shown to be secure.

The point $\mathsf{H}(\mathsf{aid})$ is used in computing $\mathsf{Sig}_1 = Y_1^x \cdot \mathsf{H}(\mathsf{aid})^r$. We can use the API provided EC key generation algorithm to generate r as the secret key and $\mathsf{H}(\mathsf{aid})^r$ as the public key. The benefit of computing $\mathsf{H}(\mathsf{aid})^r$ this way is that the algorithm checks if the point $\mathsf{H}(\mathsf{aid})$ is actually on the curve and returns an error if it is not. The only way the helper device can abuse this is by sending $-y$ instead of the correct y. This would mean the card would return $\mathsf{Sig}_1 = Y_1^x \cdot \mathsf{H}(\mathsf{aid})^{-r}$. However, such a value can be easily obtained by the helper device by computing $(\mathsf{Sig}_1, \mathsf{Sig}_2^{-1}, \mathsf{Sig}_3^{-1})$ and therefore gives no additional advantage.

It remains to show how to compute Sig_1 using the key Y_1^x (stored on the card as an EC point). To do this we use our custom implementation of point addition. To make this operation more efficient we only compute the x-coordinate of the result and let the helper device recompute y and $-y$. This saves us some operation in $\mathbb{F}_p$ on-card and the helper device can easily find the correct value using the SFPK verification procedure.

5.3 Results

Various smart cards differ in computational power and available algorithms, which influences the efficiency of custom cryptographic algorithms. Thus, a comparison with results in related work would not present meaningful data about the efficiency. However, an easy way to assess the efficiency is to compare the algorithms execution time to other well-known cryptographic algorithms. In Table 3 we compare our implementation of CObtain/CShow with elliptic curve DSA, Diffie-Hellman, and key generation algorithms. All algorithms are provided by the Multos API and work on the used smart card. Additionally, we provide a prototype implementation of the FIDO ECDAA algorithm [24, Chapter 3.5.2]. Note that the efficiency of q-SDH based DAA schemes referenced in Table 1 are close. This is due to the same number of point multiplications which is the dominant computational factor. The execution time of our ECDAA implementation can be used as a good estimator of the execution time of the other algorithms in Table 1.

The numbers given in Table 3 correspond to an average of 100 executions. It is easy to see that our algorithms are roughly two times slower than securely generating an elliptic curve key pair on-card which is one of the basic operations used in practice. A ECDAA implementation is two times slower than the smart card part of our scheme. What is more, even a full showing of credentials for CHAC is faster than just the smart card part of ECDAA.

To perform a comprehensive evaluation we created a simple android application that naively implements the algorithms used by the helper device and verifier. The core bilinear group operations were implemented using the Java based bnpairings library [5]. The only optimization used was the quaternary window method for point multiplication with pre-computation. We used pre-computation for group generators g_1, g_2 and the core device's SFPK public key which is the same for each invocation of HObtain/HShow.

Algorithm	Time
ECDSA	150
ECDH	210
ECKeyGen	222
CObtain/CShow	468
ECDAA [24]	970

On-card execution time

Algorithm	PC	Phone
HObtain	7	93
HShow	15	189
Verify	140	1003
Verify*	109	945
Issue	156	-

cred **with** 10 **Attributes**

Algorithm	PC	Phone
HObtain	7	93
HShow	15	190
Verify	200	1770
Verify*	109	954
Issue	1024	-

cred **with** 100 **Attributes**

Algorithm	PC	Phone
HObtain	7	93
HShow	15	192
Verify	851	9363
Verify*	110	960
Issue	10047	-

cred **with** 1000 **Attributes**

Table 3: Average execution time in milliseconds for BN-256 curve (N = 100). Worst case scenario for all algorithms. Bilinear pairings implemented using bnpairings **Java library based on** BigIntegers**. In algorithm** Verify* **we assume that the verifier uses pre-computed values $H_{\mathsf{Attr}}(v_{\mathsf{Attr}}) \in \mathbb{G}_2$.**

Data type	Size: bits	Size: group elements
areq - credential request	1536	$4 \cdot [\mathbb{G}_1]+[\mathbb{G}_2]$
asig - show token	3072	$6 \cdot [\mathbb{G}_1]+3 \cdot [\mathbb{G}_2]$
cred - credential	$L \cdot 1536$	$2L \cdot [\mathbb{G}_1]+2L \cdot [\mathbb{G}_2]$
apreq - partial request	1792	$4 \cdot [\mathbb{G}_1]+[\mathbb{G}_2] + [\mathbb{Z}_p]$
apsig - partial token	1792	$4 \cdot [\mathbb{G}_1]+[\mathbb{G}_2] + [\mathbb{Z}_p]$

Table 4: Size of data types for credential cred **with L attributes. Bit size is presented for the BN-256 curve.**

In our implementation, we used the standard Java based SHA-256 to implement the used pseudo-random function and for hashing to both curves, where we assume that the system is setup in a way that the hashed values always correspond to a x-coordinate on the curve. This is similar to the hash to point function that we introduced for the smart card implementation. We executed the same code on a PC (laptop) with Intel i7-7660U CPU @ 2.50 GHz with 16GB RAM. We also implemented the algorithm used by the issuer. For showing a credential we consider the worst-case scenario which for our construction is showing all attributes in a given credential.

The results are given in Table 3. It is easy to see that our construction is practical, since proving possession of even 1000 attributes takes around 0.5s in case the helper device is a PC and 0.7s in case a smartphone is used. Since we use a Java implementation for bilinear pairing this is a pessimistic estimate and a native ARM library will significantly increase efficiency on the smartphone. Show token verification is heavily influenced by our implementation of hashing to $\mathbb{G}_2$. In case the values $H_{\mathsf{Attr}}(v_{\mathsf{Attr}})$ are pre-computed, verifying takes almost the same amount of time for all sizes of credentials. This is not an impractical assumption since the number of attributes and values for an application must be limited. Otherwise, if values are unique the credential becomes traceable. The most time-consuming operation is the Issue algorithm. Fortunately, this workload can be distributed since is consists of generating AAEQ signatures on the same message but with different secret keys.

Finally, in Table 4 we present the size for credential requests, show tokens and credentials stored by the helper device. We will use $[\mathbb{Z}_p]$, $[\mathbb{G}_1]$ and $[\mathbb{G}_2]$ to respectively denote the element sizes and s is used to denote the number of attributes in a credential.

6 DISCUSSION AND FURTHER EXTENSIONS

In this section we discuss certain extensions and properties of our construction.

Optional revocation. Contrary to some previous AC models and constructions, in our CHAC model we do not consider revocation. But we will show how to extend our generic construction from Section 4 to allow blacklisting of core devices, i.e., revoke credentials corresponding to a given device.

Recall that the core device uses the SFPK.KeyGen algorithm to generate SFPK keys. For revocation we can replace it with the trapdoor generation SFPK.TKGen algorithm that outputs keys with the same distribution and additionally a trapdoor δ_{SFPK} that can be used in the SFPK.ChkRep algorithm. The core device can share this trapdoor with the helper device since this does not break unforgeability. The helper device can encrypt it with respect to the authorities' public key and use standard zero-knowledge (ZK) proofs to prove that the ciphertext contains δ_{SFPK} for which $\mathsf{SFPK.ChkRep}(\delta_{\mathsf{SFPK}}, \mathsf{pk}_{\mathsf{SFPK}}) = 1$. Note that in our instantiation this corresponds to checking pairing product equations for which we know efficient non-interactive ZK proofs [47].

Finally, once a device is blacklisted the revocation authority can decrypt and publish the trapdoor, which can be used by verifiers to check if the current session corresponds to revoked credentials. This approach obviously discloses all the show tokens (past and future) created by the revoked device. A more general approach that prevents this is as follows. Instead of the trapdoor δ_{SFPK}, we publish a randomized SFPK public key pk_i^R of the i-th blacklisted device. Now in addition to a show token $\mathsf{asig} = (\mathsf{pk}'_{\mathsf{SFPK}}, \mathsf{Sig}'_{\mathsf{SFPK}}, \sigma'_{\mathsf{Attr}})$ the helper creates a ZK proof that there exists a trapdoor δ_{SFPK} for which $\mathsf{SFPK.ChkRep}(\delta_{\mathsf{SFPK}}, \mathsf{pk}'_{\mathsf{SFPK}}) = 1$ and $\mathsf{SFPK.ChkRep}(\delta_{\mathsf{SFPK}}, \mathsf{pk}_i^R) = 0$ for all pk_i^R on the blacklist.

Pre-loading credentials. In our model, we assume that credentials are used for systems where the helper device is also part of the user's platform. However, this is not the case for some applications like for example e-tickets where the terminal that communicates with the smart card (i.e., core device) is part of the service.

A solution for this setting is to pre-randomize the SFPK public key and the AAEQ signatures by the helper device and store them on the core device. To show such a credential, the core device can simply sign the aid for the given session nonce and use the stored values to create the full asig. Due to the memory constrains of the core device, this however only works when the helper is frequently available and the user can simply re-load "fresh" values.

Pre-randomized values can only be used by the core device because of the dependability property. Thus they can be stored in an online database where each entry will be associated with a unique identifier that is generated by the helper device. To allow the core device to recompute those identifiers the helper device creates them by hashing a secret key k_{pre} together with a counter.

Distributed/Parallel issuing. An interesting property of our construction is that the issuing algorithm can be easily distributed between different servers (representing the issuing authority). Recall that for each attribute the respective AAEQ secret key $\mathsf{sk}_{\mathsf{Attr}_i} \leftarrow \mathsf{AAEQ.AKGen}(\mathsf{msk}, \mathsf{Attr}_i)$ is used to sign the SFPK public key that is part of the credential request. The resulting credential is just a tuple that contains all the AAEQ signatures on the SFPK public key for each attribute. An easy way to distribute the workload is as follows. Each server receives a dedicated set of attributes and the corresponding AAEQ secret key. Once a request is received and verified it is sent to the responsible servers which compute the AAEQ signature and return them to a server combining the results.

(Un)Trusted setup. Our generic construction from Section 4.2 uses a trusted setup to generate a common reference string (CRS) ρ. This is only required if the used SFPK scheme needs a CRS, as it is the case for our instantiation. In particular, the CRS in Scheme 1 is composed of BG and two values $Y_1 = g_1^y$ and $Y_2 = g_2^y$. The group parameters can be easily computed using a deterministic procedure and without secret coins, as it is the case for BN curves [4]. Unfortunately, this is not the case for Y_1 and Y_2. It is required that the value y is unknown, otherwise, the SFPK scheme is forgeable. On the bright side, knowing y does not help in breaking the class-hiding property which is used to ensure the unlinkability of credentials.

A simple corollary from the above discussion is that in case the system consists only of one issuer the CRS can be generated by that entity. Unfortunately, it is not possible in case of multiple issuers as the knowledge of y would allow using credentials of users issued by different issuers. A workaround would be to generate an additive share between all issuers. Instead of using values $Y_{1,i}$ and $Y_{2,i}$ generated by the i-th issuer, the CRS is constructed as $Y_1 = \prod_{i=1}^{n} Y_{1,i}$, $Y_2 = \prod_{i=1}^{n} Y_{2,i}$ where we use shares of each of the n issuers. Note that this is a well-known technique and involves additional step, i.e., a proof of knowledge of the shared discrete logarithm.

Acknowledgements. We thank Fabian Eidens, Octavio Perez Kempner and the anonymous reviewers for their helpful feedback. Lucjan Hanzlik was supported by the German Federal Ministry of Education and Research (BMBF) through funding for CISPA and the CISPA-Stanford Center for Cybersecurity (FKZ: 16KIS0762). Daniel Slamanig was supported by the European commission through ECSEL Joint Undertaking (JU) under grant agreement n°826610 (Comp4Drones) and by the Austrian Science Fund (FWF) and netidee SCIENCE under grant agreement P31621-N38 (Profet).

REFERENCES

[1] Michael Backes, Lucjan Hanzlik, Kamil Kluczniak, and Jonas Schneider. 2018. Signatures with Flexible Public Key: Introducing Equivalence Classes for Public Keys. In *ASIACRYPT 2018, Part II (LNCS, Vol. 11273).* 405–434.
[2] Michael Backes, Lucjan Hanzlik, and Jonas Schneider-Bensch. 2019. Membership Privacy for Fully Dynamic Group Signatures. In *ACM CCS 2019.* 2181–2198.
[3] Foteini Baldimtsi and Anna Lysyanskaya. 2013. Anonymous credentials light. In *ACM CCS 2013.* 1087–1098.
[4] Paulo S. L. M. Barreto and Michael Naehrig. 2006. Pairing-Friendly Elliptic Curves of Prime Order. In *SAC 2005 (LNCS, Vol. 3897).* 319–331.
[5] Paulo S. L. M. Barreto and Geovandro C. C. F. Pereira. 2015. Barreto-Naehrig (BN) pairing-friendly elliptic curves. https://github.com/javabeanz/bnpairings.

[6] Lejla Batina, Jaap-Henk Hoepman, Bart Jacobs, Wojciech Mostowski, and Pim Vullers. 2010. Developing Efficient Blinded Attribute Certificates on Smart Cards via Pairings. In *CARDIS 2010*, Dieter Gollmann, Jean-Louis Lanet, and Julien Iguchi-Cartigny (Eds.). Springer.

[7] Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. 2009. Randomizable Proofs and Delegatable Anonymous Credentials. In *CRYPTO 2009 (LNCS, Vol. 5677)*. 108–125.

[8] Mihir Bellare and Phillip Rogaway. 1993. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *ACM CCS 93*. 62–73.

[9] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. 2011. High-Speed High-Security Signatures. In *CHES 2011 (LNCS, Vol. 6917)*. 124–142.

[10] Patrik Bichsel, Jan Camenisch, Thomas Groß, and Victor Shoup. 2009. Anonymous credentials on a standard java card. In *ACM CCS 2009*. 600–610.

[11] Ronny Bjones, Ioannis Krontiris, Pascal Paillier, and Kai Rannenberg. 2012. Integrating Anonymous Credentials with eIDs for Privacy-Respecting Online Authentication. In *APF 2012*. Springer.

[12] Marina Blanton. 2008. Online subscriptions with anonymous access. In *ASIACCS 08*. 217–227.

[13] Johannes Blömer and Jan Bobolz. 2018. Delegatable Attribute-Based Anonymous Credentials from Dynamically Malleable Signatures. In *ACNS 18 (LNCS, Vol. 10892)*. 221–239.

[14] Johannes Blömer, Jan Bobolz, Denis Diemert, and Fabian Eidens. 2019. Updatable Anonymous Credentials and Applications to Incentive Systems. In *ACM CCS 2019*. 1671–1685.

[15] Jan Bobolz, Fabian Eidens, Stephan Krenn, Daniel Slamanig, and Christoph Striecks. 2020. Privacy-Preserving Incentive Systems with Highly Efficient Point-Collection. In *ASIACCS 20*. 319–333.

[16] Dan Boneh, Ben Lynn, and Hovav Shacham. 2001. Short Signatures from the Weil Pairing. In *ASIACRYPT 2001 (LNCS, Vol. 2248)*. 514–532.

[17] Dan Boneh and Victor Shoup. 2020. *A Graduate Course in Applied Cryptography (version 0.5)*. cryptobook.us.

[18] Stefan Brands. 2002. A technical overview of digital credentials. *Available online, Feb* 20 (2002), 145–8.

[19] Ernie Brickell and Jiangtao Li. 2012. Enhanced Privacy ID: A Direct Anonymous Attestation Scheme with Enhanced Revocation Capabilities. *IEEE Trans. Dependable Secur. Comput.* 9, 3 (2012), 345–360.

[20] Ernest F. Brickell, Jan Camenisch, and Liqun Chen. 2004. Direct Anonymous Attestation. In *ACM CCS 2004*. 132–145.

[21] Jan Camenisch. 2006. Protecting (Anonymous) Credentials with the Trusted Computing Group's TPM V1.2. In *(SEC 2006)*. Springer.

[22] Jan Camenisch, Liqun Chen, Manu Drijvers, Anja Lehmann, David Novick, and Rainer Urian. 2017. One TPM to Bind Them All: Fixing TPM 2.0 for Provably Secure Anonymous Attestation. In *2017 IEEE Symposium on Security and Privacy*. 901–920.

[23] Jan Camenisch, Manu Drijvers, Petr Dzurenda, and Jan Hajny. 2019. Fast Keyed-Verification Anonymous Credentials on Standard Smart Cards. In *SEC 2019*, Gurpreet Dhillon, Fredrik Karlsson, Karin Hedström, and André Zúquete (Eds.). Springer.

[24] Jan Camenisch, Manu Drijvers, Alec Edgington, Anja Lehmann, and Rainer Urian. 2018. FIDO ECDAA Algorithm. https://fidoalliance.org/specs/fido-v2.0-id-20180227/fido-ecdaa-algorithm-v2.0-id-20180227.html.

[25] Jan Camenisch, Manu Drijvers, and Anja Lehmann. 2016. Anonymous Attestation Using the Strong Diffie Hellman Assumption Revisited. In *TRUST 2016*. Springer.

[26] Jan Camenisch, Manu Drijvers, and Anja Lehmann. 2016. Universally Composable Direct Anonymous Attestation. In *PKC 2016, Part II (LNCS, Vol. 9615)*. 234–264.

[27] Jan Camenisch, Maria Dubovitskaya, Kristiyan Haralambiev, and Markulf Kohlweiss. 2015. Composable and Modular Anonymous Credentials: Definitions and Practical Constructions. In *ASIACRYPT 2015, Part II (LNCS, Vol. 9453)*. 262–288.

[28] Jan Camenisch and Anna Lysyanskaya. 2001. An Efficient System for Non-transferable Anonymous Credentials with Optional Anonymity Revocation. In *EUROCRYPT 2001 (LNCS, Vol. 2045)*. 93–118.

[29] Jan Camenisch and Anna Lysyanskaya. 2004. Signature Schemes and Anonymous Credentials from Bilinear Maps. In *CRYPTO 2004 (LNCS, Vol. 3152)*. 56–72.

[30] Jan Camenisch and Els Van Herreweghen. 2002. Design and Implementation of The Idemix Anonymous Credential System. In *ACM CCS 2002*. 21–30.

[31] Melissa Chase, Sarah Meiklejohn, and Greg Zaverucha. 2014. Algebraic MACs and Keyed-Verification Anonymous Credentials. In *ACM CCS 2014*. 1205–1216.

[32] Melissa Chase, Trevor Perrin, and Greg Zaverucha. 2020. The Signal Private Group System and Anonymous Credentials Supporting Efficient Verifiable Encryption. In *ACM CCS 20*. 1445–1459.

[33] David Chaum. 1982. Blind Signatures for Untraceable Payments. In *CRYPTO'82*. 199–203.

[34] David Chaum and Eugène van Heyst. 1991. Group Signatures. In *EUROCRYPT'91 (LNCS, Vol. 547)*. 257–265.

[35] Liqun Chen and Rainer Urian. 2015. DAA-A: Direct Anonymous Attestation with Attributes. In *TRUST 2015*.

[36] Scott E. Coull, Matthew Green, and Susan Hohenberger. 2009. Controlling Access to an Oblivious Database Using Stateful Anonymous Credentials. In *PKC 2009 (LNCS, Vol. 5443)*. 501–520.

[37] Geoffroy Couteau and Michael Reichle. 2019. Non-interactive Keyed-Verification Anonymous Credentials. In *PKC 2019, Part I (LNCS, Vol. 11442)*. 66–96.

[38] Elizabeth C. Crites and Anna Lysyanskaya. 2019. Delegatable Anonymous Credentials from Mercurial Signatures. In *CT-RSA 2019 (LNCS, Vol. 11405)*. 535–555.

[39] Elizabeth C. Crites and Anna Lysyanskaya. 2020. Mercurial Signatures for Variable-Length Messages. Cryptology ePrint Archive, Report 2020/979. https://eprint.iacr.org/2020/979.

[40] Fergus Dall, Gabrielle De Micheli, Thomas Eisenbarth, Daniel Genkin, Nadia Heninger, Ahmad Moghimi, and Yuval Yarom. 2018. CacheQuote: Efficiently Recovering Long-term Secrets of SGX EPID via Cache Attacks. *IACR TCHES* 2018, 2 (2018), 171–191. https://tches.iacr.org/index.php/TCHES/article/view/879.

[41] Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda. 2018. Privacy Pass: Bypassing Internet Challenges Anonymously. *PoPETs* 2018, 3 (2018), 164–180.

[42] Dominic Deuber, Matteo Maffei, Giulio Malavolta, Max Rabkin, Dominique Schröder, and Mark Simkin. 2018. Functional Credentials. *PoPETs* 2018, 2 (April 2018), 64–84.

[43] Georg Fuchsbauer and Romain Gay. 2018. Weakly Secure Equivalence-Class Signatures from Standard Assumptions. In *PKC 2018, Part II (LNCS, Vol. 10770)*. 153–183.

[44] Georg Fuchsbauer, Christian Hanser, and Daniel Slamanig. 2015. Practical Round-Optimal Blind Signatures in the Standard Model. In *CRYPTO 2015, Part II (LNCS, Vol. 9216)*. 233–253.

[45] Georg Fuchsbauer, Christian Hanser, and Daniel Slamanig. 2019. Structure-Preserving Signatures on Equivalence Classes and Constant-Size Anonymous Credentials. *Journal of Cryptology* 32, 2 (April 2019), 498–546.

[46] Christina Garman, Matthew Green, and Ian Miers. 2014. Decentralized Anonymous Credentials. In *NDSS 2014*.

[47] Jens Groth and Amit Sahai. 2008. Efficient Non-interactive Proof Systems for Bilinear Groups. In *EUROCRYPT 2008 (LNCS, Vol. 4965)*. 415–432.

[48] Christian Hanser and Daniel Slamanig. 2014. Structure-Preserving Signatures on Equivalence Classes and Their Application to Anonymous Credentials. In *ASIACRYPT 2014, Part I (LNCS, Vol. 8873)*. 491–511.

[49] Chloé Hébant and David Pointcheval. 2020. Traceable Constant-Size Multi-Authority Credentials. Cryptology ePrint Archive, Report 2020/657. https://eprint.iacr.org/2020/657.

[50] Thomas S. Heydt-Benjamin, Hee-Jin Chae, Benessa Defend, and Kevin Fu. 2006. Privacy for Public Transportation. In *PET 2006 (LNCS, Vol. 4258)*. 1–19.

[51] Thomas Icart. 2009. How to Hash into Elliptic Curves. In *CRYPTO 2009 (LNCS, Vol. 5677)*. 303–316.

[52] Jonathan Katz and Nan Wang. 2003. Efficiency Improvements for Signature Schemes with Tight Security Reductions. In *ACM CCS 2003*. 155–164.

[53] Mojtaba Khalili, Daniel Slamanig, and Mohammad Dakhilalian. 2019. Structure-Preserving Signatures on Equivalence Classes from Standard Assumptions. In *ASIACRYPT 2019, Part III (LNCS, Vol. 11923)*. 63–93.

[54] Armen Khatchatourov, Maryline Laurent, and Claire Levallois-Barth. 2015. Privacy in Digital Identity Systems: Models, Assessment, and User Adoption. In *Electronic Government*, Efthimios Tambouris, Marijn Janssen, Hans Jochen Scholl, Maria A. Wimmer, Konstantinos Tarabanis, Mila Gascó, Bram Klievink, Ida Lindgren, and Peter Parycek (Eds.). Springer International Publishing, Cham, 273–290.

[55] Stephan Krenn, Thomas Lorünser, Anja Salzer, and Christoph Striecks. 2017. Towards Attribute-Based Credentials in the Cloud. In *CANS 17 (LNCS, Vol. 11261)*. 179–202.

[56] Ben Kreuter, Tancrède Lepoint, Michele Orrù, and Mariana Raykova. 2020. Anonymous Tokens with Private Metadata Bit. In *CRYPTO 2020, Part I (LNCS, Vol. 12170)*. 308–336.

[57] Michael Z. Lee, Alan M. Dunn, Brent Waters, Emmett Witchel, and Jonathan Katz. 2013. Anon-Pass: Practical Anonymous Subscriptions. In *2013 IEEE Symposium on Security and Privacy*. 319–333.

[58] Benoît Libert, Fabrice Mouhartem, Thomas Peters, and Moti Yung. 2016. Practical "Signatures with Efficient Protocols" from Simple Assumptions. In *ASIACCS 16*. 511–522.

[59] Jinyu Lu, Yunwen Liu, Tomer Ashur, Bing Sun, and Chao Li. 2020. Rotational-XOR Cryptanalysis of Simon-Like Block Ciphers. In *ACISP 20 (LNCS, Vol. 12248)*. 105–124.

[60] MAOSCO Limited. 2020. MULTOS Standard Technology. https://www.multos.com/.

[61] Milica Milutinovic, Koen Decroix, Vincent Naessens, and Bart De Decker. 2015. Privacy-Preserving Public Transport Ticketing System. In *Data and Applications Security and Privacy XXIX*. Springer.

[62] Daniel Moghimi, Berk Sunar, Thomas Eisenbarth, and Nadia Heninger. 2020. TPM-FAIL: TPM meets Timing and Lattice Attacks. In *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, Srdjan Capkun and Franziska Roesner (Eds.). USENIX Association, 2057–2073. https://www.usenix.org/conference/usenixsecurity20/presentation/moghimi-tpm

[63] Wojciech Mostowski and Pim Vullers. 2012. Efficient U-Prove Implementation for Anonymous Credentials on Smart Cards. In *Security and Privacy in Communication Networks*, Muttukrishnan Rajarajan, Fred Piper, Haining Wang, and George Kesidis (Eds.). Springer.

[64] Oracle. 2020. Java Card Technology. https://www.oracle.com/java/technologies/java-card-tech.html.

[65] Christian Paquin and Greg Zaverucha. 2013. U-Prove Cryptographic Specification V1.1 (Revision 3). https://www.microsoft.com/en-us/research/publication/u-prove-cryptographic-specification-v1-1-revision-3/

[66] David Pointcheval and Olivier Sanders. 2016. Short Randomizable Signatures. In *CT-RSA 2016 (LNCS, Vol. 9610)*. 111–126.

[67] Himanshu Raj, Stefan Saroiu, Alec Wolman, Ronald Aigner, Jeremiah Cox, Paul England, Chris Fenner, Kinshuman Kinshumann, Jork Löser, Dennis Mattoon, Magnus Nyström, David Robinson, Rob Spiger, Stefan Thom, and David Wooten. 2016. fTPM: A Software-Only Implementation of a TPM Chip. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, Thorsten Holz and Stefan Savage (Eds.). USENIX Association, 841–856. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/raj

[68] Kai Rannenberg, Jan Camenisch, and Ahmad Sabouri (Eds.). 2015. *Attribute-based Credentials for Trust: Identity in the Information Society*. Springer.

[69] Olivier Sanders. 2020. Efficient Redactable Signature and Application to Anonymous Credentials. In *PKC 2020, Part II (LNCS, Vol. 12111)*. 628–656.

[70] Michael Schwarz and Daniel Gruss. 2020. How Trusted Execution Environments Fuel Research on Microarchitectural Attacks. *IEEE Secur. Priv.* 18, 5 (2020), 18–27. https://doi.org/10.1109/MSEC.2020.2993896

[71] Adi Shamir. 1984. Identity-Based Cryptosystems and Signature Schemes. In *CRYPTO'84 (LNCS, Vol. 196)*. 47–53.

[72] Victor Shoup. 1997. Lower Bounds for Discrete Logarithms and Related Problems. In *EUROCRYPT'97 (LNCS, Vol. 1233)*. 256–266.

[73] Alberto Sonnino, Mustafa Al-Bassam, Shehar Bano, Sarah Meiklejohn, and George Danezis. 2019. Coconut: Threshold Issuance Selective Disclosure Credentials with Applications to Distributed Ledgers. In *NDSS 2019*.

[74] Eric R. Verheul. 2001. Self-Blindable Credential Certificates from the Weil Pairing. In *ASIACRYPT 2001 (LNCS, Vol. 2248)*. 533–551.

[75] Brent R. Waters. 2005. Efficient Identity-Based Encryption Without Random Oracles. In *EUROCRYPT 2005 (LNCS, Vol. 3494)*. 114–127.

A OMITTED FORMAL DEFINITIONS

A.1 Preliminaries

We write $\mathsf{Exp}^{\phi}_{\mathcal{A},\Psi}(1^\lambda) \Rightarrow 1$ for the event that the experiment Exp returns 1, when instantiated with parameters ϕ, adversary $\mathcal{A}$ and primitive Ψ, all of which possibly omitted. We define the *adjusted advantage* of adversary $\mathcal{A}$ in this experiment as

$$\mathbf{Adv}\,[x]\;{}^{\mathsf{Exp}^{\phi}}_{\mathcal{A},\Psi}(1^\lambda) := \left|\Pr\left[\mathsf{Exp}^{\phi}_{\mathcal{A},\Psi}(1^\lambda) \Rightarrow 1\right] - x\right|$$

If $x = 0$, we write instead $\mathbf{Adv}^{\mathsf{Exp}^{\phi}}_{\mathcal{A},\Psi}(1^\lambda)$ for its *advantage*.

A.2 Signatures with Flexible Public Key

Definition A.1 (Class-hiding with Key Corruption). For SFPK with relation $\mathcal{R}$ and adversary $\mathcal{A}$ we define the following experiment:

```
$\mathsf{C\text{-}H}^{\mathcal{R}}_{\mathcal{A},\mathsf{SFPK}}(\lambda)$
$(\mathsf{sk}_i, \mathsf{pk}_i) \xleftarrow{\$} \mathsf{SFPK.KeyGen}(1^\lambda)$ for $i \in \{0,1\}$
$b \xleftarrow{\$} \{0,1\}; r \xleftarrow{\$} \mathsf{coin}$
$(\mathsf{sk}', \mathsf{pk}') \leftarrow \mathsf{SFPK.ChgKeys}(\mathsf{sk}_b, \mathsf{pk}_b, r)$
$\hat{b} \xleftarrow{\$} \mathcal{A}^{\mathsf{SFPK.Sign}(\mathsf{sk}',\cdot)}((\mathsf{sk}_0,\mathsf{pk}_0),(\mathsf{sk}_1,\mathsf{pk}_1),\mathsf{pk}')$
return $b = \hat{b}$
```

A SFPK is *class-hiding with key corruption* if for all *PPT* adversaries $\mathcal{A}$, their advantage $\mathbf{Adv}\left[\frac{1}{2}\right]{}^{\mathsf{C\text{-}H}}_{\mathcal{A},\mathsf{SFPK}}(1^\lambda)$ is negligible.

Definition A.2 (Existential Unforgeability under Flexible Public Key). For scheme SFPK with relation $\mathcal{R}$ and adversary $\mathcal{A}$ we define the following experiment:

```
$\mathsf{EUF\text{-}CMA}^{\mathcal{R}}_{\mathcal{A},\mathsf{SFPK}}(\lambda)$
$(\mathsf{sk}, \mathsf{pk}, \delta) \xleftarrow{\$} \mathsf{SFPK.TKGen}(1^\lambda); Q := \emptyset$
$(\mathsf{pk}', m^*, \mathsf{Sig}^*) \xleftarrow{\$} \mathcal{A}^{O_1(\mathsf{sk},\cdot), O_2(\mathsf{sk},\cdot,\cdot)}(\mathsf{pk}, \delta)$
return $m^* \notin Q \wedge$
 return $\mathsf{SFPK.ChkRep}(\delta, \mathsf{pk}') = 1 \wedge$
        $\mathsf{SFPK.Verify}(\mathsf{pk}', m^*, \mathsf{Sig}^*) = 1$
```

```
$O_1(\mathsf{sk}, m)$
$\mathsf{Sig} \xleftarrow{\$} \mathsf{SFPK.Sign}(\mathsf{sk}, m)$
$Q := Q \cup \{m\}$
return Sig

$O_2(\mathsf{sk}, m, r)$
$\mathsf{sk}' \xleftarrow{\$} \mathsf{SFPK.ChgSK}(\mathsf{sk}, r)$
$\mathsf{Sig} \xleftarrow{\$} \mathsf{SFPK.Sign}(\mathsf{sk}', m)$
$Q := Q \cup \{m\}$
return Sig
```

A SFPK is *existentially unforgeable with flexible public key under chosen message attacks* if for all *PPT* adversaries $\mathcal{A}$, their advantage $\mathbf{Adv}^{\mathsf{EUF\text{-}CMA}^{\mathcal{R}}}_{\mathcal{A},\mathsf{SFPK}}(1^\lambda)$ is negligible.

A.3 Structure Preserving Signatures on Equivalence Classes

EUF-CMA security is similar to that of conventional signatures, but a forgery needs to be with respect to an unqueried class.

Definition A.3 (EUF-CMA). For scheme SPS-EQ and adversary $\mathcal{A}$ we define the following experiment:

```
$\mathsf{EUF\text{-}CMA}_{\mathcal{A},\mathsf{SPS\text{-}EQ}}(\lambda, \ell)$
$\mathsf{BG} \xleftarrow{\$} \mathsf{Setup}(\lambda); Q := \emptyset$
$(\mathsf{sk}, \mathsf{pk}) \xleftarrow{\$} \mathsf{KeyGen}(\mathsf{BG}, \ell)$
$(M^*, \sigma^*) \leftarrow \mathcal{A}^{O_1(\mathsf{sk},\cdot)}(\mathsf{pk})$
return $[M^*] \neq [M]\ \forall M \in Q \wedge$
       $\mathsf{Verify}(\mathsf{pk}, M^*, \sigma^*) = 1$

$O_1(\mathsf{sk}, M)$
$\sigma \xleftarrow{\$} \mathsf{Sign}(\mathsf{sk}, M)$
$Q := Q \cup \{M\}$
return $\sigma$
```

An SPS-EQ over $(\mathbb{G}_i^*)^\ell$ is existentially unforgeable under adaptively chosen-message attacks, if for all PPT adversaries $\mathcal{A}$, their advantage $\mathbf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{A},\mathsf{SPS\text{-}EQ}}(1^\lambda, \ell)$ is negligible.

Definition A.4 (Perfect Adaption of Signatures under malicious keys [44]). Let $\ell > 1$. An SPS-EQ scheme SPS-EQ on $(\mathbb{G}_i^*)^\ell$ perfectly adapts signatures under malicious keys if for all tuples $(\mathsf{pk}, M, \sigma, \mu)$ with $M \in (\mathbb{G}_i^*)^\ell \;\wedge\; \mathsf{Verify}(M, \sigma, \mathsf{pk}) = 1 \;\wedge\; \mu \in \mathbb{Z}_p^*$ we have that the output of $\mathsf{ChgRep}(M, \sigma, \mu, \mathsf{pk})$ is a uniformly random element in the space of signatures, conditioned on $\mathsf{Verify}(M^\mu, \sigma', \mathsf{pk}) = 1$.

A relaxation of this definition (perfect adaption) considers tuples of the form $(\mathsf{sk}, \mathsf{pk}, M, \sigma, \mu)$ for which $\mathsf{VKey}(\mathsf{sk}, \mathsf{pk}) = 1$ and requires that the output of $\mathsf{ChgRep}(M, \sigma, \mu, \mathsf{pk})$ and $\mathsf{Sign}(M^\mu, \mathsf{sk})$ are identically distributed. We note that for our CHAC construction we only need this relaxed definition.

A.4 Tag-Based Equivalence Class Signatures

Definition A.5 (EUF-CMA). For scheme TBEQ and adversary $\mathcal{A}$ we define the following experiment:

```
$\mathsf{EUF\text{-}CMA}_{\mathcal{A},\mathsf{TBEQ}}(\lambda, \ell)$
$\mathsf{pars} \xleftarrow{\$} \mathsf{Setup}(\lambda); Q := \emptyset$
$(\mathsf{sk}, \mathsf{pk}) \xleftarrow{\$} \mathsf{KeyGen}(\mathsf{pars}, \ell)$
$(M^*, \sigma^*, \tau^*) \leftarrow \mathcal{A}^{O_1(\mathsf{sk},\cdot,\cdot)}_{\mathsf{CMA}}(\mathsf{pk})$
return $\mathsf{Verify}(\mathsf{pk}, M^*, \tau^*, \sigma^*) = 1 \wedge$
    $([M^*], \tau^*) \neq ([M], \tau)\ \forall (M, \tau) \in Q$
```

```
$O_1(\mathsf{sk}, M, \tau)$
$\sigma \xleftarrow{\$} \mathsf{Sign}(\mathsf{sk}, M, \tau)$
$Q := Q \cup \{(M, \tau)\}$
return $\sigma$
```

A TBEQ is EUF-CMA, secure if for all PPT adversaries $\mathcal{A}$, their advantage $\mathbf{Adv}^{\mathsf{EUF}}_{\mathcal{A},\mathsf{TBEQ}}(1^\lambda, \ell)$ is negligible.

A.5 Aggregatable Attribute-Based EQs

Definition A.6 (EUF-CMA). For scheme AAEQ and adversary $\mathcal{A}$ we define the following experiment:

```
$\mathsf{EUF\text{-}CMA}_{\mathcal{A},\mathsf{AAEQ}}(\lambda, t, \ell)$
$(\mathsf{msk}, \mathsf{mpk}) \xleftarrow{\$} \mathsf{Setup}(1^\lambda, t, \ell); Q, A := \emptyset$
$(M^*, \sigma^*, \{\mathsf{Attr}^*_i\}) \xleftarrow{\$} \mathcal{A}^{O_1(\mathsf{msk},\cdot,\cdot,\cdot)}(\mathsf{mpk})$
return $\bigwedge_i (\mathsf{Attr}^*_i, v^*_{\mathsf{Attr}_i}, [M^*]) \notin Q \wedge$
        $\mathsf{Verify}(\mathsf{mpk}, \{\mathsf{Attr}^*_i\}, M^*, \sigma^*) = 1$
```

```
$O_1(\mathsf{msk}, \mathsf{Attr}, v_{\mathsf{Attr}}, M)$
if $(\mathsf{Attr}, \cdot) \notin A$
    $\mathsf{sk}_{\mathsf{Attr}} \xleftarrow{\$} \mathsf{AKGen}(\mathsf{msk}, \mathsf{Attr})$
    $A := A \cup \{(\mathsf{Attr}, \mathsf{sk}_{\mathsf{Attr}})\}$
$\sigma \xleftarrow{\$} \mathsf{Sign}(\mathsf{sk}_{\mathsf{Attr}}, v_{\mathsf{Attr}}, M)$
$Q := Q \cup \{(\mathsf{Attr}, v_{\mathsf{Attr}}, M)\}$
return $\{\sigma\}$
```

An AAEQ is *existentially unforgeable under chosen message attacks* if for all *PPT* adversaries $\mathcal{A}$, the advantage $\mathbf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{A},\mathsf{AAEQ}}(1^\lambda, t, \ell)$ is negligible.

Definition A.7 (Perfect Adaption of Signatures). An AAEQ scheme on $(\mathbb{G}^*_i)^\ell$ perfectly adapts signatures if for all tuples $(\{\mathsf{sk}_{\mathsf{Attr}_i}\}, \mathsf{mpk}, M, \{\mathsf{Attr}_i\}, \sigma, \mu)$ where it holds that $\mathsf{VKey}(\{\mathsf{sk}_{\mathsf{Attr}_i}\}, \mathsf{mpk}) = 1$, $\mathsf{Verify}(\mathsf{mpk}, \{\mathsf{Attr}_i\}, \sigma, M) = 1$, $M \in (\mathbb{G}^*_i)^\ell$, and $\mu \in \mathbb{Z}^*_p$, the distributions $(M^\mu, \mathsf{Agg}(\mathsf{mpk}, \{\mathsf{Sign}(\mathsf{sk}_{\mathsf{Attr}_i}, v_{\mathsf{Attr}_i}, M^\mu,)\})$ and $\mathsf{ChgRep}(M, \sigma, \mu, \mathsf{mpk})$ are identical.

A.6 CHAC: Formal Model

Let HD, CD, SN, MN be empty sets. We introduce lists DSK, $CRED$, $ATTR$, D, AID, I2D to track honest device secret keys, credentials issued to honest devices, the corresponding attributes, device identifiers, session identifiers for issuing/showing, a list used to identify which credential corresponds to which honest device. Additionally, we will use an array $CATTR$ to store sets with attributes of dishonest devices where we use the device identifiers as indexes to the array. Finally, we introduce a counter c_{AID} initialized to 0. Moreover, let us define the following oracles.

$O_{HD}(i)$: takes as input an identifier i and outputs $\perp$ if $i \in HD \cup CD$. Otherwise, it creates a honest core device by running $DSK[i] \xleftarrow{\$} \mathsf{CKGen}(1^\lambda)$, adding i to HD and setting $D[i] = \perp$.

$O_{\mathsf{nonce}}()$: this allows the adversary to initiate an issuing/showing session. The oracle chooses nonce $\xleftarrow{\$} \{0,1\}^\lambda$, increments counter c_{AID} and sets $AID[c_{AID}] = \mathsf{nonce}$. Finally, it returns $(c_{AID}, \mathsf{nonce})$.

$O_{\mathsf{ObtIss}}(i, \mathsf{Attr})$: creates credentials for honest device i, i.e. it outputs $\perp$ if $i \notin HD$. Otherwise, it generates a nonce nonce $\xleftarrow{\$} \{0,1\}^\lambda$, generates aid $\xleftarrow{\$} \mathsf{AIDGen}(\mathsf{Attr}, \mathsf{nonce})$ and issues a credential for i by running apreq $\xleftarrow{\$} \mathsf{CObtain}(\mathsf{aid}, \mathsf{ipk}, DSK[i])$, areq $\xleftarrow{\$} \mathsf{HObtain}(\mathsf{Attr}, \mathsf{nonce}, \mathsf{ipk}, \mathsf{apreq})$, and $(\mathsf{cred}, \mathsf{did}) \xleftarrow{\$} \mathsf{Issue}(\mathsf{Attr}, \mathsf{nonce}, \mathsf{areq}, \mathsf{isk})$. If cred $= \perp$ it returns $\perp$. Otherwise it adds $(i, \mathsf{cred}, \mathsf{Attr})$ to lists (I2D, $CRED$, $ATTR$) and sets $D[i] = \mathsf{did}$.

$O_{CD}(i)$: takes as input an identifier i. If $i \notin HD$ it outputs $\perp$. Otherwise, it creates a corrupted core device by adding i to CD and setting $HD = HD \setminus \{i\}$. If $D[i] \neq \perp$ it computes the union $CATTR[D[i]]$ of all sets $ATTR[j]$ for all j where $\mathsf{I2D}[j] = i$. Finally, it returns $DSK[i]$.

$O_{\mathsf{Issue}}(s, \mathsf{Attr}, \mathsf{areq})$: allows the adversary, who impersonates a malicious device, to obtain credentials. It takes as input a session index $s > 0$ and returns $\perp$ if $AID[j] = \perp$. The oracle generates $(\mathsf{cred}, \mathsf{did}) \xleftarrow{\$} \mathsf{Issue}(\mathsf{Attr}, AID[j], \mathsf{areq}, \mathsf{isk})$ and aborts if cred $= \perp$. Otherwise, it computes the union $CATTR[\mathsf{did}] = CATTR[\mathsf{did}] \cup \mathsf{Attr}$. The oracle sets $AID[j] = \perp$ and returns cred.

$O_{\mathsf{CShow}}(i, \mathsf{aid})$: allows the adversary to obtain a partial show tokens from an honest device and impersonate a malicious helper device. It takes a input a device index i and attribute identifier aid. If $i \notin HD$ then return $\perp$. Otherwise, compute apsig $\xleftarrow{\$} \mathsf{CShow}(\mathsf{aid}, \mathsf{ipk}, DSK[i])$, adds (aid) to set SN and return apsig.

$O_{\mathsf{HShow}}(j, \mathsf{nonce}, \mathsf{Attr})$: allows the adversary, who impersonates a malicious verifier, to trigger showings with an honest device. It takes as input an index of an issuance j, nonce and a set of attributes Attr. Let $i \leftarrow \mathsf{I2D}[j]$. If $i \notin HD$ or $\mathsf{Attr} \not\subseteq ATTR[j]$ or $CRED[j] = \perp$ then return $\perp$. Otherwise, compute aid $\leftarrow \mathsf{AIDGen}(\mathsf{Attr}, \mathsf{nonce})$, apsig $\xleftarrow{\$} \mathsf{CShow}(\mathsf{aid}, \mathsf{ipk}, DSK[i])$ and asig $\xleftarrow{\$} \mathsf{HShow}(\mathsf{Attr}, \mathsf{nonce}, CRED[j], \mathsf{ipk}, \mathsf{apsig})$. Add (nonce) to MN and return asig.

$O_{\mathsf{Obtain}_1}(i, \mathsf{Attr}, \mathsf{nonce})$: allows the adversary, who impersonates a malicious issuer, to issue credentials for a honest device. It takes as input a device index i and returns $\perp$ if $i \notin HD$. Otherwise it computes aid $\xleftarrow{\$} \mathsf{AIDGen}(\mathsf{Attr}, \mathsf{nonce})$, apreq $\xleftarrow{\$} \mathsf{CObtain}(\mathsf{aid}, \mathsf{ipk}, DSK[i])$, and areq $\xleftarrow{\$} \mathsf{HObtain}(\mathsf{Attr}, \mathsf{nonce}, \mathsf{ipk}, \mathsf{apreq})$. and adds $(i, \varepsilon, \mathsf{Attr})$ to lists (I2D, $CRED$, $ATTR$).

$O_{\mathsf{Obtain}_2}(j, \mathsf{cred})$: allows the adversary, who impersonates a malicious issuer, to issue credentials for a honest device. It takes as input a device index j and returns $\perp$ if cred $= \perp$ or $CRED[j] \neq \varepsilon$. Otherwise, it sets $CRED[j] = \mathsf{cred}$.

We define correctness, compactness, unforgeability, dependability and anonymity as the following experiments. We assume that, if required, the experiment honestly generates a reference string ρ using $\mathsf{Setup}(1^\lambda)$ which is an implicit argument for the remaining algorithms.

Definition A.8 (Correctness). A core/helper anonymous credentials system is *correct* if for all $\lambda \in \mathbb{N}$, all key pairs $(\mathsf{isk}, \mathsf{ipk}) \xleftarrow{\$} \mathsf{IKGen}(1^\lambda)$, all secret key ssk $\xleftarrow{\$} \mathsf{CKGen}(1^\lambda)$, all attribute sets $\mathsf{Attr}_s \subseteq \mathsf{Attr}_o$ and all nonces $\mathsf{nonce}_o, \mathsf{nonce}_s \in \{0,1\}^\lambda$, $\mathsf{aid}_o \xleftarrow{\$} \mathsf{AIDGen}(\mathsf{Attr}_o, \mathsf{nonce}_o)$, $\mathsf{aid}_s \xleftarrow{\$} \mathsf{AIDGen}(\mathsf{Attr}_s, \mathsf{nonce}_s)$, all credential requests areq $\xleftarrow{\$} \mathsf{HObtain}(\mathsf{Attr}_o, \mathsf{nonce}_o, \mathsf{ipk}, \mathsf{CObtain}(\mathsf{aid}_o,$

ipk, ssk)), all showings asig $\xleftarrow{\$}$ HShow(Attr$_s$, nonce$_s$, cred, CShow(aid$_s$, ssk)), we have Verify(Attr, nonce, areq, ipk) = 1, where (cred, did) $\xleftarrow{\$}$ Issue(Attr, nonce$_o$, asig, isk).

Definition A.9 (Compactness). A core/helper anonymous credentials system is *compact* if for all $\lambda \in \mathbb{N}$, all key pairs (isk, ipk) $\xleftarrow{\$}$ IKGen(1^λ), all secret key ssk $\xleftarrow{\$}$ CKGen(1^λ), all attribute sets Attr$_s$ $\subseteq$ Attr$_o$ and all nonces nonce$_o$, nonce$_s$ $\in \{0,1\}^\lambda$, aid$_o$ $\xleftarrow{\$}$ AIDGen(Attr$_o$, nonce$_o$), aid$_s$ $\xleftarrow{\$}$ AIDGen(Attr$_s$, nonce$_s$), all credential requests areq $\xleftarrow{\$}$ HObtain(Attr$_o$, nonce$_o$, ipk, CObtain(aid$_o$, ipk, ssk)), all showings asig $\xleftarrow{\$}$ HShow(Attr$_s$, nonce$_s$, cred, CShow(aid$_s$, ssk)), we have |asig| $\leq O(\lambda)$, i.e., the size of the showing token asig is independent of the attribute set |Attr$_s$| and only depends on λ.

Definition A.10 (Unforgeability). For the core/helper anonymous credential and adversary $\mathcal{A}$ we define the following experiment:

```
$\mathsf{UNF}^{\mathcal{A}}_{\mathsf{CHAC}}(\lambda)$
$(\mathsf{isk}, \mathsf{ipk}) \xleftarrow{\$} \mathsf{IKGen}(1^\lambda)$
$\mathsf{nonce} \xleftarrow{\$} \{0,1\}^\lambda$
$O := \{O_{HD}, O_{CD}, O_{\mathsf{nonce}}, O_{\mathsf{ObtIss}}, O_{\mathsf{Issue}}, O_{\mathsf{HShow}}\}$
$(\mathsf{Attr}^*, \mathsf{asig}^*) \xleftarrow{\$} \mathcal{A}^O(\mathsf{ipk}, \mathsf{nonce})$
if $\mathsf{Verify}(\mathsf{Attr}^*, \mathsf{nonce}, \mathsf{asig}^*, \mathsf{ipk}) = 1$ and $\forall_j\ \mathsf{Attr}^* \not\subseteq CATTR[j]$
 and $(\mathsf{nonce}) \notin \mathsf{MN}$ then return 1
else return 0
```

A CHAC is *unforgeable* if for all *PPT* adversaries $\mathcal{A}$, its advantage in the above experiment is negligible:

$$\mathsf{Adv}^{\mathsf{unf}}_{\mathcal{A},\mathsf{CHAC}}(\lambda) = \Pr\Big[\mathsf{UNF}^{\mathcal{A}}_{\mathsf{CHAC}}(\lambda) = 1\Big] = \mathsf{negl}(\lambda).$$

Definition A.11 (Dependability). For the core/helper anonymous credential and adversary $\mathcal{A}$ we define the following experiment:

```
$\mathsf{DEP}^{\mathcal{A}}_{\mathsf{CHAC}}(\lambda)$
$(\mathsf{isk}, \mathsf{ipk}) \xleftarrow{\$} \mathsf{IKGen}(1^\lambda)$
$O := \{O^{(1)}_{HD}, O_{\mathsf{ObtIss}}, O_{\mathsf{nonce}}, O_{\mathsf{Issue}}, O_{\mathsf{CShow}}\}$
$(\mathsf{Attr}^*, \mathsf{nonce}^*, \mathsf{asig}^*) \xleftarrow{\$} \mathcal{A}^O(\mathsf{ipk})$
$\mathsf{aid}^* \xleftarrow{\$} \mathsf{AIDGen}(\mathsf{Attr}^*, \mathsf{nonce}^*)$
if $(\mathsf{aid}^*) \in \mathsf{SN}$ then return 0
if $\mathsf{Verify}(\mathsf{Attr}^*, \mathsf{nonce}^*, \mathsf{asig}^*, \mathsf{ipk}) = 1$ and
$\forall_j\ \mathsf{Attr}^* \not\subseteq CATTR[j]$ then
    return 1
else return 0
```

A CHAC is *dependable* if for all *PPT* adversaries $\mathcal{A}$, its advantage in the above experiment is negligible:

$$\mathsf{Adv}^{\mathsf{dep}}_{\mathcal{A},\mathsf{CHAC}}(\lambda) = \Pr\Big[\mathsf{DEP}^{\mathcal{A}}_{\mathsf{CHAC}}(\lambda) = 1\Big] = \mathsf{negl}(\lambda).$$

Definition A.12 (Anonymity). For the core/helper anonymous credential and adversary $\mathcal{A}$ we define the following experiment:

```
$\mathsf{ANON}^{\mathcal{A}}_{\mathsf{CHAC}}(\lambda)$
$b \xleftarrow{\$} \{0,1\}$
$O := \{O_{HD}, O_{CD}, O_{\mathsf{Obtain}_1}, O_{\mathsf{Obtain}_2}, O_{\mathsf{HShow}}\}$
$(j_0, j_1, \mathsf{Attr}^*, \mathsf{nonce}^*, \mathsf{isk}^*, \mathsf{ipk}^*, \mathsf{st}) \xleftarrow{\$} \mathcal{A}^O(\lambda)$
$i_0 \xleftarrow{\$} \mathsf{I2D}[j_0];\ i_1 \xleftarrow{\$} \mathsf{I2D}[j_1]$
if $i_0, i_1 \notin HD$ or $\mathsf{Attr}^* \not\subseteq ATTR[j_0] \cap ATTR[j_1]$ then return 0
$\mathsf{aid}^* \xleftarrow{\$} \mathsf{AIDGen}(\mathsf{Attr}^*, \mathsf{nonce}^*)$
$\mathsf{apsig} \xleftarrow{\$} \mathsf{CShow}(\mathsf{aid}^*, \mathsf{ipk}^*, DSK[i_b])$
$\mathsf{asig} \xleftarrow{\$} \mathsf{HShow}(\mathsf{Attr}^*, \mathsf{nonce}^*, CRED[j_b], \mathsf{ipk}^*, \mathsf{apsig})$
$b^* \xleftarrow{\$} \mathcal{A}^O(\mathsf{asig}, \mathsf{st})$
return $b^* = b$
```

A CHAC is *anonymous* if for all *PPT* adversaries $\mathcal{A}$, its advantage in the above experiment is negligible:

$$\mathsf{Adv}^{\mathsf{anon}}_{\mathcal{A},\mathsf{CHAC}}(\lambda) = \Pr\Big[\mathsf{ANON}^{\mathcal{A}}_{\mathsf{CHAC}}(\lambda) = 1\Big] = \mathsf{negl}(\lambda).$$

Note that the adversary returns isk* which means that in our definition we assume an honestly generated issuer's key. This can be ensured using standard proof techniques, i.e. the issuer proves knowledge of the secret key. We define anonymity this way to simplify our construction and proofs.

B PROOFS FOR SECTION 3

B.1 Proof of Theorem 3.2

Proof. Let $(\mathsf{BG}, g_1^a, g_1^b, g_1^c, g_1^d, g_2^a, g_2^b, g_2^c, g_2^d)$ be an instance of the BDDH problem. We will show that we can use any efficient adversary $\mathcal{A}$ to solve the above problem instance. To do so, we will build a reduction algorithm $\mathcal{R}$ that uses $\mathcal{A}$ in a black box manner.

Let q_h the maximal number of random oracle queries made by the adversary $\mathcal{A}$ and $(\mathsf{Sig}^*_{\mathsf{SFPK}}, m^*, \mathsf{pk}^*_{\mathsf{SFPK}})$ be the forgery returned by an adversary $\mathcal{A}$, where $\mathsf{Sig}^*_{\mathsf{SFPK}} = (\mathsf{Sig}^*_1, \mathsf{Sig}^*_2, \mathsf{Sig}^*_3)$. The reduction choose a random index $i \in \{1, \ldots, q_h\}$ and aborts the experiment in case m^* is not the i-th query of $\mathcal{A}$ to the random oracle. Note that this means that the probability that $\mathcal{R}$ does not abort the experiment at any point is $1/q_h$. What is more, for the i-th random oracle query $\mathsf{H}(m^*)$ the reduction answers with $g_1^{h_{m^*}}$.

To simulate the unforgeabilty experiment, the reduction first prepares the common reference string ρ by setting $Y_1 = g_1^a, Y_2 = g_2^a$. Next $\mathcal{R}$ prepares the public key $\mathsf{pk}_{\mathsf{SFPK}}$ and the trapdoor τ_{SFPK}. For this it uses the values g_1^b and g_2^b from the problem instance. It sets $\mathsf{pk}_{\mathsf{SFPK}} = (g_1, g_1^b)$ and $\tau_{\mathsf{SFPK}} = (g_2^b)$. Moreover, the reduction chooses $k_u \xleftarrow{\$} \mathbb{Z}_p^*$, sets $\mathsf{st}_{\mathsf{pub}} = ((g_1^a) \cdot g_1^{k_u}, (g_2^a) \cdot g_2^{k_u})$ and shares it with $\mathcal{A}$.

To answer $\mathcal{A}$'s signing queries for message m and randomness t (which is equal to 1 for oracle O^1), the reduction $\mathcal{R}$ follows the following steps:

(1) it first chooses $w_t \xleftarrow{\$} \mathbb{Z}_p^*$,
(2) it programs the random oracle to output $\mathsf{H}(m) = (g_1^b)^{-w_t^{-1}} \cdot g_1^{h_m}$ for some $h_m \xleftarrow{\$} \mathbb{Z}_p^*$,
(3) compute $w = w_t \cdot t$,
(4) it computes: $\mathsf{Sig}^1_{\mathsf{SFPK}} = (g_1^b)^{t \cdot k_u} \cdot (U_1^w)^{h_m}$,
(5) set the pre-signature $\mathsf{pSig}_{\mathsf{SFPK}} := (\mathsf{Sig}^1_{\mathsf{SFPK}}, w)$.

It is easy to see that this is a valid pre-signature. Note that a valid one is of the form $(g_1^{a\cdot b\cdot t}\cdot((g_1^b)^{-w^{-1}}\cdot g_1^{h_m})^r, w)$. In this case, the reduction has set $r = t\cdot w\cdot(a+k_u)$ and this means that the $g_1^{a\cdot b\cdot t}$ cancels out and the reduction does not need to compute $g_1^{a\cdot b}$. Note that this only works because the reduction is able to program the random oracle and does not actually know the value r. We also assume that if $\mathcal{A}$ queries a message m prior to a query to signing queries, the reduction answers with $\mathsf{H}(m) = (g_1^b)^{-w^{-1}}\cdot g_1^{h_m}$ and retains (w, h_m).

Finally, the adversary outputs the forgery $(\mathsf{pk}^*_{\mathsf{SFPK}}, m^*, \mathsf{Sig}^*_{\mathsf{SFPK}})$ of $\mathcal{A}$ and the reduction proceeds as follows:

(1) parse $\mathsf{Sig}^*_{\mathsf{SFPK}}$ as $(\mathsf{Sig}^1_{\mathsf{SFPK}}, \mathsf{Sig}^2_{\mathsf{SFPK}}, \mathsf{Sig}^3_{\mathsf{SFPK}})$,

(2) compute
$$\begin{aligned} g_1^{a\cdot b\cdot t^*} &= \mathsf{Sig}^1_{\mathsf{SFPK}}\cdot(\mathsf{Sig}^2_{\mathsf{SFPK}})^{-h_{m^*}} \\ &= \left(g_1^{a\cdot b\cdot t^*}\cdot \mathsf{H}(m^*)^{r^*}\cdot(g_1^{r^*})^{-h_{m^*}}\right), \\ &= \left(g_1^{a\cdot b\cdot t^*}\cdot (g_1^{h_m})^{r^*}\cdot(g_1^{r^*})^{-h_{m^*}}\right), \end{aligned}$$

(3) parse $\mathsf{pk}^*_{\mathsf{SFPK}}$, and since for a valid forgery then $\mathsf{pk}^*_{\mathsf{SFPK}} \in [\mathsf{pk}_{\mathsf{SFPK}}]_{\mathcal{R}}$ and we have $\mathsf{pk}^*_{\mathsf{SFPK}} = (g_1^{t^*}, (g_1^b)^{t^*})$ and $\mathcal{R}$ can use $g_1^{t^*}$,

(4) output 1 iff $e(g_1^{a\cdot b\cdot t^*}, g_2^c) = e(g_1^{t^*}, g_2^d)$.

The probability that $\mathcal{R}$ successfully solves the bilinear decisional Diffie-Hellman problem depends on the advantage of $\mathcal{A}$ and the probability that $\mathcal{R}$'s simulation succeeds.

□

B.2 Proof of Lemma 3.6

Proof. We exactly follow the proof of the underlying FHS15 SPS-EQ scheme in [45] and only highlight the differences. To ease the readability we write elements in $\mathbb{G}_2$ with "hat", e.g., as $\hat{V}$ instead of V_2, and consequently the forgery is denoted as $(Z, Y, \hat{Y}, \hat{V})$. Now, if the take the discrete logarithms of all available group elements in the forgery, we get an additional $\hat{V}^*$ term $(\hat{v}^*)$ and need to consider the contributions of the h elements (with coefficients θ_i) and $\hat{v}_j$ elements (with coefficients ν_j) from the q queries. So the changes to $\hat{y}^*$ and the additional element $\hat{v}^*$ are:

$$\hat{y}^* = \pi_{\hat{y}} + \sum_{i\in[\ell]}\chi_{\hat{y},i}x_i + \sum_{i\in[k]}\theta_{\hat{y},i}h_i + \sum_{j\in[q]}\nu_{\hat{y},j}v_j + \sum_{j\in[q]}\psi_{\hat{y},j}\frac{1}{y_j} \quad (1a)$$

$$\hat{v}^* = \pi_{\hat{v}} + \sum_{i\in[\ell]}\chi_{\hat{v},i}x_i + \sum_{i\in[k]}\theta_{\hat{v},i}h_i + \sum_{j\in[q]}\nu_{\hat{v},j}v_j + \sum_{j\in[q]}\psi_{\hat{v},j}\frac{1}{y_j} \quad (1b)$$

From the forgery we know that we have

$$\sum_{i\in[\ell]} m_i^* x_i = z^*\hat{y}^* \quad (2a)$$

$$y^* = \hat{y}^* \quad (2b)$$

$$\hat{v}^* = y^*\hat{h}^* \quad (2c)$$

We can now follow the proof for FHS15 and in particular Claim 1 and Corollary 1 (which is exactly as in their proof), and by using the same argumentation as in FHS15 for (2b), from

$$y^* = \pi_y + \sum_{j\in[q]}\rho_{y,j}z_j + \sum_{j\in[q]}\psi_{y,j}\frac{1}{y_j}$$

for (1a) we need to have $\pi_y = \pi_{\hat{y}}$ and the non-zero coefficients are $\psi_{y,j}$ and $\psi_{\hat{y},j}$, where we have $\psi_{y,j} = \psi_{\hat{y},j}$ for all $i \in [q]$. Consequently, the proof continues exactly as the FHS15 with the only difference that we additionally need to investigate (2c). By leveraging the simplification of Eq. (9) in [45], we know that there exists one $n \in [q]$ for which $y^* = \psi_{y,n}\frac{1}{y_n}$. By construction we have $h^* = h_i$ for a given $i \in [k]$, i.e., the tag τ_i of the forgery. Now only considering non-zero coefficients we can simplify (1b) to

$$\hat{v}^* = \sum_{i\in[k]}\theta_{\hat{v},i}h_i + \sum_{j\in[q]}\nu_{\hat{v},j}v_j.$$

From FHS15 we know that $\rho_{z,j}\pi_y z_n = 0$ for all $j \in [q]$. But since z_j and $\rho_{z,j}$ are non-zero for some j, we have $\pi_y = 0$ and thus $\theta_{\hat{v},i} = 0$ for all $i \in [k]$. By equating coefficients we have

$$y^*h_i = \psi_{y,n}\frac{1}{y_n}h_i \text{ and } \hat{v}^* = \sum_{j\in[q]}\nu_{\hat{v},j}(h_i\frac{1}{y_i}).$$

By leveraging the fact that all y_i are distinct, we obtain that $\hat{v}^* = \nu_{\hat{v},n}(h_i\frac{1}{y_n})$ with $\nu_{\hat{v},n} = \psi_{y,n}$ yielding that the $\hat{v}^*$ part is consistent with the remainder representing a previous query with the exact same tag and in particular the entire forgery is just a multiple of previously queried message. Note that the simulation error is the same as in the FHS15 proof.

□

B.3 Proof of Lemma 3.8

Proof. For perfect adaption under malicious keys let $M \in (\mathbb{G}_1^*)^\ell$, $\tau \in \{0,1\}^*$, $H : \{0,1\}^* \rightarrow \mathbb{G}_2$, $\mathsf{pk} \in (\mathbb{G}_2^*)^\ell$ and $(x_i)_{i\in[\ell]}$ be such that $\mathsf{pk} = (g_2^{x_i})_{i\in[\ell]}$. A signature $(Z_1, Y_1, Y_2, V_2) \in \mathbb{G}_1\times\mathbb{G}_1^*\times\mathbb{G}_2^*\times\mathbb{G}_2$ satisfying $\mathsf{Verify}(M, (Z_1, Y_1, Y_2, V_2), \mathsf{pk}) = 1$ is of the form $((\prod(M_i^{x_i})^y, g_1^{\frac{1}{y}}, g_2^{\frac{1}{y}}, H(\tau)^{\frac{1}{y}})$ for some $y \in \mathbb{Z}_p$. $\mathsf{ChgRep}(M, (Z_1, Y_1, Y_2, V_2), \mu, \mathsf{pk})$ for $\mu \in \mathbb{Z}_p$ outputs $((\prod(M_i^{x_i})^{y\psi}, g_1^{\frac{1}{y\psi}}, g_2^{\frac{1}{y\psi}}, H(\tau)^{\frac{1}{y\psi}})$, which is a uniformly random element σ in the signature space conditioned on $\mathsf{Verify}(M^\mu, \sigma, \mathsf{pk}) = 1$.

TBEQ in Scheme 2 also satisfies the conventional perfect adaption notion, since $\mathsf{sk} = (x_i)_{i\in[\ell]}$ is the only element satisfying $\mathsf{VKey}(\mathsf{sk}, \mathsf{pk}) = 1$ (which checks if $\mathsf{pk} = g_2^{\mathsf{sk}}$) and $\mathsf{Sign}(M^\mu, \mathsf{sk})$ (as ChgRep) outputs a uniformly random element σ in the space of signatures conditioned on $\mathsf{Verify}(M^\mu, \sigma, \mathsf{pk}) = 1$.

□

B.4 Proof of Lemma 3.12

We use the same notation as in the proof of Lemma 3.6 and note that we consider a bounded attribute-space represented by distinct and random elements $h_i \in \mathbb{G}_2^*$, $i \in [k]$, in the mpk (i.e., one for every possible $(\mathsf{Attr}, v_{\mathsf{Attr}})$ pair). Moreover, for the sake of readability we prove the lemma for the case $t = 2$ with public keys $\mathsf{pk}_1 = (g_2^{x_i})_{i\in[\ell]}$ and $\mathsf{pk}_2 = (g_2^{u_i})_{i\in[\ell]}$ respectively and it is straightforward to generalize it to any $n > 2$. Note that a query for the same representative M to either of the keys results in using the same randomness y. We require that for any message M to the Sign oracle of AAEQ, if the adversary wants to obtain a signature for more than one attribute, it will obtain signatures under both secret keys (attributes) using the same randomness y (which is sampled uniformly at random in each query to Sign) and the queried attribute values v_{Attr} and $v_{\mathsf{Attr}'}$ (which maps to two of the h values). We will denote the corresponding Z_1 elements of signatures under pk_1 and pk_2 using superscript (1) and (2) respectively.

As in the proof of Lemma 3.6, we follow the the proof of the underlying FHS15 SPS-EQ scheme in [45]. We start by taking the discrete logarithms of all elements:

$$\begin{aligned}
z^* &= \pi_z + \sum_{j\in[q]} \rho^{(1)}_{z,j} z^{(1)}_j + \sum_{j\in[q]} \rho^{(2)}_{z,j} z^{(2)}_j + \sum_{j\in[q]} \psi_{z,j} \frac{1}{y_j} \\
y^* &= \pi_y + \sum_{j\in[q]} \rho^{(1)}_{y,j} z^{(1)}_j + \sum_{j\in[q]} \rho^{(2)}_{y,j} z^{(2)}_j + \sum_{j\in[q]} \psi_{y,j} \frac{1}{y_j} \\
\hat{y}^* &= \pi_{\hat{y}} + \sum_{i\in[\ell]} \chi_{\hat{y},i} x_i + \sum_{i\in[\ell]} \omega_{\hat{y},i} u_i + \sum_{i\in[k]} \theta_{\hat{y},i} h_i \\
&\quad + \sum_{j\in[q]} \nu_{\hat{y},j} v_j + \sum_{j\in[q]} \psi_{\hat{y},j} \frac{1}{y_j} \\
\hat{v}^* &= \pi_{\hat{v}} + \sum_{i\in[\ell]} \chi_{\hat{v},i} x_i + \sum_{i\in[\ell]} \omega_{\hat{v},i} u_i + \sum_{i\in[k]} \theta_{\hat{v},i} h_i \\
&\quad + \sum_{j\in[q]} \nu_{\hat{v},j} v_j + \sum_{j\in[q]} \psi_{\hat{v},j} \frac{1}{y_j} \\
m^*_i &= \pi_{m^*,i} + \sum_{j\in[q]} \rho^{(1)}_{m^*,i,j} z^{(1)}_j + \sum_{j\in[q]} \rho^{(2)}_{m^*,i,j} z^{(2)}_j \\
&\quad + \sum_{j\in[q]} \psi_{m^*,i,j} \frac{1}{y_j} \\
m_{j,i} &= \pi_{m,j,i} + \sum_{k\in[j-1]} \rho^{(1)}_{m,j,i,k} z^{(1)}_k + \sum_{k\in[j-1]} \rho^{(2)}_{m,j,i,k} z^{(2)}_k \\
&\quad + \sum_{k\in[j-1]} \psi_{m,j,i,k} \frac{1}{y_k}
\end{aligned}$$

And from the forgery we know that:

$$\sum_{i\in[\ell]} m^*_i (x_i + u_i) = z^* \hat{y}^* \tag{3a}$$

$$y^* = \hat{y}^* \tag{3b}$$

$$\hat{v}^* = y^* (\hat{h}^*_1 + \hat{h}^*_2) \tag{3c}$$

with the pair of $(\mathsf{Attr}, v_{\mathsf{Attr}})$ values in the forgery w.l.o.g. corresponding to h_1 and h_2 respectively. In the following we omit the analysis of Equation (3c) as this follows from the exact same reasoning as in the proof of Lemma 3.6. First, we observe that we can adopt Claim 1 and Corollary 1 from the FHS15 proof in [45] to our case of the $z^{(1)}_n$ and $z^{(2)}_n$ which in particular means that all y's in such monomials are different, one is y_n and for every x as well as u there comes one y. Moreover, $z^{(1)}_n$ contains one more x than u's and vice-versa for $z^{(2)}_n$. Now, we first look at Equation (3b) and comparing coefficients immediately yields that $\pi_{y^*} = \pi_{\hat{y}^*}$, that $\chi_{\hat{y},i} = \omega_{\hat{y},i} = 0$ for all $i \in [\ell]$, $\theta_{\hat{y},i} = \nu_{\hat{v},j} = 0$ for all $i \in [k]$ and $\psi_{y,j} = \psi_{\hat{y},j}$ for all $j \in [q]$. Moreover, due to Claim 1 we have that $\rho^{(1)}_{y,j} = \rho^{(2)}_{y,j} = 0$ for all $j \in [q]$. This simplifies Equation (3b) to

$$y^* = y = \pi_y + \sum_{j\in[q]} \psi_{y,j} \frac{1}{y_j}.$$

Now, we use this simplification to investigate Equation (3a):

$$\begin{aligned}
&\sum_{i\in[\ell]} (\pi_{m^*,i} + \sum_{j\in[q]} \rho^{(1)}_{m^*,i,j} z^{(1)}_j + \sum_{j\in[q]} \rho^{(2)}_{m^*,i,j} z^{(2)}_j \\
&\quad + \sum_{j\in[q]} \psi_{m^*,i,j} \frac{1}{y_j})(x_i + u_i) = (\pi_z + \sum_{j\in[q]} \rho^{(1)}_{z,j} z^{(1)}_j \\
&\quad + \sum_{j\in[q]} \rho^{(2)}_{z,j} z^{(2)}_j + \sum_{j\in[q]} \psi_{z,j} \frac{1}{y_j})(\pi_y + \sum_{j\in[q]} \psi_{y,j} \frac{1}{y_j})
\end{aligned} \tag{4}$$

Now, by expanding the RHS and comparing coefficients it follows that $\pi_z \pi_y = 0$, $\pi_z \psi_{y,j} = 0$, $\pi_y \psi_{z,j} = 0$, $\pi_y \rho^{(b)}_{z,j} = 0$ for all $j \in [q]$, $b \in [2]$ and $\psi_{z,j} \psi_{y,k} = 0$ for all $j, k \in [q]$. This simplifies the RHS to:

$$\sum_{j\in[q]} \sum_{k\in[q]} \rho^{(1)}_{z,j} \psi_{y,k} z^{(1)}_j \frac{1}{y_k} + \sum_{j\in[q]} \sum_{k\in[q]} \rho^{(2)}_{z,j} \psi_{y,k} z^{(2)}_j \frac{1}{y_k} \tag{5}$$

Now, we take a closer look at Equation (5) and Claim 1 tells us that every $z^{(b)}_j$, $b \in [2]$, has an equal number of y's and x's (respectively u's) in the numerator and consequently for all monomials on the LHS there is one y less than x's (or u's respectively). Consequently, following the same argumentation as in [45] we obtain that $\rho^{(1)}_{z,j} \psi_{y,k} = 0$ and $\rho^{(2)}_{z,j} \psi_{y,k} = 0$ for all $j \neq k$ (note that it may be the case that either of $z^{(1)}$ or $z^{(2)}$ may not be present at all, but one needs to be non-zero to represent a valid forgery. We will consider the case where both are present subsequently, the other cases are analogous). Furthermore, following the FHS15 argumentation it follows that there is exactly one $n \in [q]$ s.t. $\rho^{(b)}_{z,n} \psi_{y,n} \neq 0$. Consequently, we obtain a simplified version of Equation (5) as

$$\rho^{(1)}_{z,n} \psi_{y,n} z^{(1)}_n \frac{1}{y_n} + \rho^{(2)}_{z,n} \psi_{y,n} z^{(2)}_n \frac{1}{y_n}$$

and substituting $z^{(b)}_n$ by its definition and simplification we obtain

$$\begin{aligned}
&\rho^{(1)}_{z,n} \psi_{y,n} \sum_{i\in[\ell]} m_{n,i} x_i + \rho^{(2)}_{z,n} \psi_{y,n} \sum_{i\in[\ell]} m_{n,i} u_i = \\
&\psi_{y,n} (\rho^{(1)}_{z,n} + \rho^{(2)}_{z,n}) \sum_{i\in[\ell]} m_{n,i} (x_i + u_i)
\end{aligned}$$

Now, plugging in $m_{n,i}$ and setting $\alpha = \psi_{y,n} (\rho^{(1)}_{z,n} + \rho^{(2)}_{z,n})$ we obtain:

$$\begin{aligned}
&\alpha(\sum_{i\in[\ell]} \pi_{m,n,i} + \sum_{k\in[j-1]} \rho^{(1)}_{m,n,i,k} z^{(1)}_k + \\
&\sum_{k\in[j-1]} \rho^{(2)}_{m,n,i,k} z^{(2)}_k + \sum_{k\in[j-1]} \psi_{m,n,i,k} \frac{1}{y_k})(x_i + u_i)
\end{aligned}$$

and by equating coefficients with the LHS of Equation (4) we obtain that $\pi_{m^*,i} = \alpha \pi_{m,n,i}$, $\rho^{(1)}_{m^*,i,j} = \alpha \rho^{(1)}_{m,n,i,k}$, $\rho^{(2)}_{m^*,i,j} = \alpha \rho^{(2)}_{m,n,i,k}$ and $\psi_{m^*,i,j} = \alpha \psi_{m,n,i,k}$, whereas the forgery just represents a previously queried message. Finally, the simulation error of the generic group is identical to FHS15.

C PROOFS FOR SECTION 4

C.1 Proof of Theorem 4.2

We will prove this theorem using a series of hybrid arguments. Let $\mathsf{asig}^* = (\mathsf{pk}^*_{\mathsf{SFPK}}, \mathsf{Sig}^*_{\mathsf{SFPK}}, \sigma^*_{\mathsf{Attr}})$ and Attr^* be the values returned by the adversary and nonce^* be the value given to the adversary.

Moreover, let q_{HD} denote the maximum number of queries made to the HD oracle by the adversary and $\mathsf{aid}^* = \mathsf{AIDGen}(\mathsf{Attr}^*, \mathsf{nonce}^*)$.

$\mathcal{H}_0$: This is the anonymity experiment.

$\mathcal{H}_1$: We change the way we generate the keys inside the $O_{HD}(i)$ oracle. Instead of SFPK.KeyGen we use trapdoor generation SFPK.TKGen and retain the trapdoor δ_i.

$\mathcal{H}_2$: We abort the experiment if there is a collision for aid^*, i.e. if there was a query for a tuple $(\mathsf{Attr}, \mathsf{nonce}) \neq (\mathsf{Attr}^*, \mathsf{nonce}^*)$ for which $\mathsf{aid}^* = \mathsf{AIDGen}(\mathsf{Attr}, \mathsf{nonce})$.

$\mathcal{H}_3$: We abort the experiment if $\mathsf{SFPK.ChkRep}(\delta_j, \mathsf{pk}^*_{\mathsf{SFPK}}) = 0$ for all $j \in [q_{HD}]$ and $j \in HD$, i.e. we do not abort if the SFPK public key is in a relation with an honest device public key.

$\mathcal{H}_4$: We choose an index $j \in [q_{HD}]$ and we abort the experiment if $\mathsf{SFPK.ChkRep}(\delta_j, \mathsf{pk}^*_{\mathsf{SFPK}}) = 0$, i.e. we chose a specific honest device.

Lemma C.1. *Hybrids $\mathcal{H}_0$ and $\mathcal{H}_1$ are indistinguishable.*

Proof. For the SFPK scheme we have that SFPK.KeyGen and SFPK.TKGen produce key pairs with identical distribution. □

Lemma C.2. *The changes made in hybrid $\mathcal{H}_2$ lowers the adversaries advantage in the unforgeability experiment only by a negligible fraction which is at most the advantage of breaking collision-resistance of* AIDGen.

Lemma C.3. *The changes made in hybrid $\mathcal{H}_3$ lowers the adversaries advantage in the unforgeability experiment only by a negligible fraction which is at most the advantage of an adversary breaking the unforgeability of the* AAEQ *scheme.*

Proof. We will show this proof via a simple reduction. The idea for the reduction is to instead of using the AKGen and Sign algorithm inside Issue to generate credentials cred for devices the reduction will use its AAEQ signing query. In the end, the adversary returns $(\mathsf{Attr}^*, \mathsf{asig}^* = (\mathsf{pk}^*_{\mathsf{SFPK}}, \mathsf{Sig}^*_{\mathsf{SFPK}}, \sigma^*_{\mathsf{Attr}}))$ which contains a AAEQ forgery for message $(\mathsf{pk}^*_{\mathsf{SFPK}}, \mathsf{Attr}^*)$.

Note that because we only abort if the SFPK public key $\mathsf{pk}_{\mathsf{SFPK}}$ is not in a relation with any of the honest device and by definition this excludes the usage of all corrupted attribute. Thus, we know that Attr^* was never queried together with an element from the class $[\mathsf{pk}^*_{\mathsf{SFPK}}]_{\mathcal{R}}$ to the AAEQ signing oracle. □

Lemma C.4. *Hybrid $\mathcal{H}_4$ does not abort with prob. $1/q_{HD}$.*

Lemma C.5. *An adversary that has non-negligible advantage against the unforgeability experiment in $\mathcal{H}_4$ can be used to break the unforgeability of the* SFPK *scheme.*

Proof. We will show this proof via a simple reduction. The idea is for the reduction to simulate the j-the device using the SFPK signing oracle. In other words, instead of running algorithm CObtain, CShow for the secret device key $DSK[j]$, the reduction asks the oracle for the corresponding signature.

Finally the adversary output asig^* for which we know that $\mathsf{SFPK.ChkRep}(\delta_j, \mathsf{pk}^*_{\mathsf{SFPK}}) = 1$, i.e. that the signature $\mathsf{Sig}^*_{\mathsf{SFPK}}$ corresponds to the device that the reduction simulated using the SFPK challenges. Thus by returning $(\mathsf{pk}^*_{\mathsf{SFPK}}, \mathsf{aid}^*, \mathsf{Sig}^*_{\mathsf{SFPK}})$ the reduction outputs a valid forgery against the SFPK unforgeability experiment. □

C.2 Proof of Theorem 4.3

We will prove this theorem using a series of hybrid arguments. Let q_{HD} denote the maximum number of queries made to the HD oracle by the adversary. Let $\mathsf{asig} = (\mathsf{pk}'_{\mathsf{SFPK}}, \mathsf{Sig}'_{\mathsf{SFPK}}, \sigma'_{\mathsf{Attr}})$ be the challenge signature given to the adversary.

$\mathcal{H}_0$: This is the anonymity experiment.

$\mathcal{H}_1$: We change the way the value σ'_{Attr} is computed inside oracle O_{HShow}, i.e. instead of randomizing the AAEQ signature using ChgRep, we use the secret key isk^* to generate a fresh signature on $\mathsf{pk}'_{\mathsf{SFPK}}$.

$\mathcal{H}_2$: We choose two distinct indexes $k_0, k_1 \in [q_{HD}]$ and abort the experiment if $i_0 \neq k_0$ and $i_1 \neq k_1$ where $i_0 \xleftarrow{\$} \mathsf{I2D}[j_0]$, $i_1 \xleftarrow{\$} \mathsf{I2D}[j_1]$ and j_0, j_1 were returned by the adversary.

Lemma C.6. *Hybrids $\mathcal{H}_0$ and $\mathcal{H}_1$ are indistinguishable assuming the* AAEQ *scheme perfectly adapts signatures.*

Lemma C.7. *The experiment is not aborted in $\mathcal{H}_2$ with probability* $(1/q_{HD})^2$.

Lemma C.8. *An adversary that has non-negligible advantage against the anonymity experiment in $\mathcal{H}_2$ can be used to break the class-hiding property of* SFPK *signatures.*

Proof. We will show this by constructing a reduction $\mathcal{R}$ which is given $((\mathsf{sk}_0, \mathsf{pk}_0), (\mathsf{sk}_1, \mathsf{pk}_1), \mathsf{pk}')$ by the challenger and access to an oracle that output valid SFPK signatures for public key pk'. The reduction uses $(\mathsf{sk}_0, \mathsf{pk}_0)$ and $(\mathsf{sk}_1, \mathsf{pk}_1)$ to respectively simulate the devices k_0 and k_1.

Finally, it receives $(j_0, j_1, \mathsf{Attr}^*, \mathsf{nonce}^*, \mathsf{isk}^*, \mathsf{ipk}^*, \mathsf{st})$ from the adversary. Because we are in $\mathcal{H}_2$ we know that j_0, j_1 correspond to devices k_0, k_1. The reduction now sets $\mathsf{pk}'_{\mathsf{SFPK}} = \mathsf{pk}'$, uses it's oracle to generate the signature $\mathsf{Sig}'_{\mathsf{SFPK}}$ on message $\mathsf{aid}^* = \mathsf{AIDGen}(\mathsf{Attr}^*, \mathsf{nonce}^*)$ and creates σ'_{Attr} as per $\mathcal{H}_2$. The adversary ends the experiment by outputting b^* which is also returned by reduction. It is easy to see that in this case $\mathsf{pk}' = \mathsf{pk}_b$ and the adversary can be used this way to break the class-hiding property. □

C.3 Proof of Theorem 4.4

The proof follows using a simple reduction. The key point to notice is that there is only one honest device created in this experiment and the reduction can use it's own signing oracle to get a SFPK signature and answer queries to the O_{CShow} oracle. What is more, since we require that $\mathsf{aid}^* \notin \mathsf{SN}$ it follows that for $\mathsf{asig}^* = (\mathsf{pk}^*_{\mathsf{SFPK}}, \mathsf{Sig}^*_{\mathsf{SFPK}}, \sigma^*_{\mathsf{Attr}})$ the tuple $(\mathsf{aid}^*, \mathsf{pk}^*_{\mathsf{SFPK}}, \mathsf{pk}^*_{\mathsf{SFPK}})$ can be used by the reduction as a valid forgery. Note that in case there exists a tuple $(\mathsf{Attr}, \mathsf{nonce}) \neq (\mathsf{Attr}^*, \mathsf{nonce}^*)$ for which $\mathsf{aid}^* = \mathsf{AIDGen}(\mathsf{Attr}, \mathsf{nonce})$ the reduction can return both pairs as a collision for AIDGen.