PAKA: Pseudonymous Authenticated Key Agreement without bilinear cryptography

PAKA: Pseudonymous Authenticated Key Agreement without bilinear cryptography

Raphael Schermann
raphael.schermann@student.tugraz.at
Institute of Technical Informatics,
Graz University of Technology
Graz, Austria

Simone Bussa
simone.bussa@polito.it
Department of Control and Computer Engineering, Politecnico di Torino
Turin, Italy

Rainer Urian
rainer.urian@infineon.com
Infineon Technologies AG
Augsburg, Germany

Ronald Toegl
ronald.toegl@infineon.com
Infineon Technologies Austria AG
Graz, Austria

Christian Steger
steger@tugraz.at
Institute of Technical Informatics,
Graz University of Technology
Graz, Austria

ABSTRACT

Anonymity and pseudonymity are important concepts in the domain of the Internet of Things. The existing privacy-preserving key agreement schemes are only concerned with maintaining the privacy of the communicated data that appears on the channel established between two honest entities. However, privacy should also include anonymity or pseudonymity of the device identity. This means there should not exist any correlation handle to associate different communications done by the device.

This paper proposes a privacy-preserving key agreement method called the Pseudoonymous Authenticated Key Agreement Protocol (PAKA), which also provides device unlinkability across different domains. This protocol is based on an Elliptic-Curve Diffie-Hellman using standard cryptographic primitives and curves, i.e., no pairing-based cryptography or other computationally intensive cryptography is necessary. For the security analysis, we provide a mathematical proof and an automatic cryptographic protocol verification utilizing Proverif. Last, we show the integration with the Trusted Platform Module and a Proof-of-Concept implementation.

CCS CONCEPTS

• **Security and privacy** → *Key management*; **Privacy-preserving protocols**; **Pseudonymity, anonymity and untraceability**; *Embedded systems security*; Hardware-based security protocols.

KEYWORDS

Pseudonymity, Privacy-preserving cryptography, Key management, TPM, Internet of Things privacy

ACM Reference Format:
Raphael Schermann, Simone Bussa, Rainer Urian, Ronald Toegl, and Christian Steger. 2024. PAKA: Pseudonymous Authenticated Key Agreement without bilinear cryptography. In *The 19th International Conference on Availability, Reliability and Security (ARES 2024), July 30–August 02, 2024, Vienna, Austria.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3664476.3669925

ARES 2024, July 30–August 02, 2024, Vienna, Austria

ACM ISBN 979-8-4007-1718-5/24/07
https://doi.org/10.1145/3664476.3669925

1 INTRODUCTION

Today, an ever-increasing number of Internet of Things (IoT) devices have become an integral part of an individual's daily routines. These devices are also linked to Cloud Computing or Fog Computing platforms [2] for data gathering and further data processing. It inherently brings challenges like heightened latency, limited power and bandwidth resources, and potential privacy risks [20]. This situation provides ample opportunities for the Service Provider to monitor users who seek to access specific services privately. Privacy remains a hot topic in the Cloud-, Fog-, and Edge Computing domains. Also, there is an exchange between two parties when it comes to key establishment. In a concrete real-world scenario, two individuals could meet and confidentially exchange keys for their locks. When it comes to the online world, there is a bit of a tricky situation that's like a chicken-and-egg dilemma. While a hybrid method for setting up and sharing keys among parties strikes a good balance between speed, security, and user experience, there still needs to be a level of trust between the parties exchanging data. Further, a crucial topic here is to preserve the privacy of those parties.

There exist several privacy-preserving algorithms, most of them are signature-based. One of the most noticeable is Elliptic Curve Direct Anonymous Attestation (ECDAA) [10, 11, 21]. This scheme uses pairing-based cryptography. Besides that, pseudonymous signatures based on standard elliptic curve cryptography exist. The most prominent approach has been defined by Bender *et al.* [5] and is used in Electronic Identification, Authentication and Trust Services (eIDAS) (see https://www.bsi.bund.de/dok/TR-03110-en).

In 2022 Schermann *et al.* [18] published a scheme that uses pairing-friendly cryptography for symmetric encryption. This scheme is called the Anonymous Authenticated Credential Key Agreement (AACKA). Nevertheless, pairing-friendly cryptography and the corresponding keys are not always supported by IoT devices and secure elements like a Trusted Platform Module. Hence, we assert the pressing need to furnish authenticated key agreement schemes

that uphold privacy without the extensive use of pairing-friendly cryptography and are particularly designed for devices with limited resources.

In both approaches, a Barreto-Naehrig curve BN256 is used. This also implies that, security-wise, the keys that we are using indicate a higher bit security. As today a NIST_P256 curve reaches 2^{128} while a BN256 curve has a 2^{100} bit security [4]. As defined by some regulators, e.g., Bundesamt für Sicherheit in der Informationstechnik (BSI) [7], the bit-security size of keys should be at least 2^{120}. Furthermore, some resource-constrained devices and secured elements like a Trusted Platform Module are yet not capable of supporting bigger BN keys, i.e., encryption schemes with, for example, a BN462 cannot be realized with a TPM.

Our contribution

We present another novel cryptographic building block called Pseudonymous Authenticated Key Agreement (PAKA) that solves the previously mentioned capability of devices that do not support pairing-based cryptography to support device pseudonymity. Our protocol emphasizes the pseudonymous authentication from a Device to a Service Provider. This means if it is used as intended, the Device appears under a chosen pseudonym and does not reveal the real identity of the Device but can be linked. The authentication process from the Service Provider to the Device can utilize any standard authentication method, as there are no specific privacy requirements for the Service Provider.

This building block can be utilized to facilitate the implementation of Pseudonymous Authenticated Encryption (PAE) across diverse application scenarios. It can be regarded as an alternative option to the AACKA protocol in case pairing-friendly cryptography is either infeasible or not the favored option for ensuring unlinkable encrypted data transmission while preserving authenticated privacy. In the end, we also show the integration of the Trusted Platform Module (TPM) into the protocol. We have decided to implement the protocol with the TPM due to its widespread adoption in IoT devices, as well as in fog and cloud computing environments. Nevertheless, the protocol can also be integrated on smaller platforms like JavaCard.

We verified the security and the related attributes with a security analysis done (1) mathematically and (2) with a formal automatic verification tool, Proverif.

Outline of the paper

The remainder of the paper looks as follows. First, we start with the necessary information regarding the background of pseudonymous signatures (section 2). We discuss the related work in section 3. This is followed by the main part of our new protocol that starts with the definition in section 4, followed by the protocol itself in section 5, including the integration in the ECIES protocol and the merge into the ECIES protocol. We also provided a security proof in section 6 to prove our protocol is still secure. The paper ends with an implementation in section 7 and a conclusion and future work (section 8).

2 BACKGROUND

In this section, we introduce the necessary background to understand the remainder of the paper. Our protocol originated from two roots: the Anonymous Authenticated Credential Key Agreement (AACKA) protocol and pseudonymous signatures. In detail, we use the idea of anonymous authenticated encryption combined with pseudonymous signatures, i.e., we do not need pairing-based cryptography. In the Background section, we cover the necessary details related to pseudonymous signatures, while the AACKA protocol will be discussed in the related work section 3.

2.1 Domain specific pseudonyms

In this subsection, we show the basic principles of domain-specific pseudonyms. Abstractly, a domain specific pseudonym can be seen as a function mapping a domain identifier and a user identifier to a unique number. The methods shown in this paper realize the mapping as elliptic curve scalar multiplications. Figure 1 shows a generic schema for pseudonymous authentication. Each user X has a private key sk_X and each service provider Y has a public key pk_Y. Each domain Y has a base name bsn_Y from which the base point $J_Y := H_{\mathbb{G}}(bsn_Y)$ is calculated.

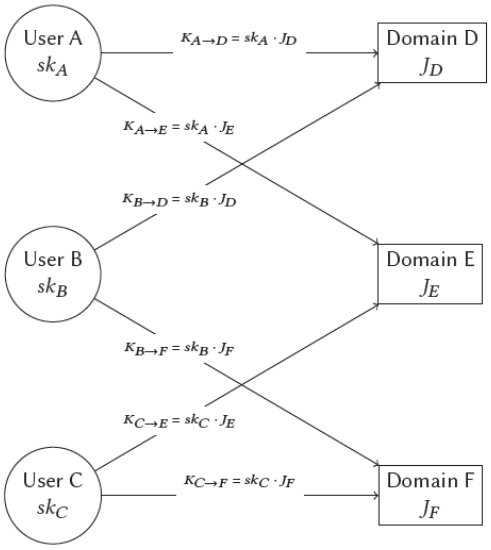


Figure 1: Basic construction of pseudonymous identifier

The domain, sometimes called in the literature sector, specific identifier K is calculated as scalar multiplication of the secret and public key, i.e., $K_{X\to Y} = sk_X * J_Y$. In the Figure, we see two users (A,B) and three domains (C,D,E). For each domain, the user chooses a different basename *bsn* that must be multiplied by the secret key. Due to the combination of the private and public keys, such a pseudonymous authentication includes a cross-sector unlinkability. This means no service knows that, for example, $K_{A\to D}$ and $K_{A\to E}$ belong to the same user A. The difference between pseudonymity and anonymity is that in the case of pseudonymity, a domain can recognize a recurring user, while in anonymity, this is not possible. However, pseudonyms belonging to different Domains, i.e., constructed by different basenames *bsn*, cannot be correlated. In the

following subsection, we focus on eIDAS pseudonymous signatures. eIDAS pseudonymous signatures can be seen as an extension of Schnorr [19] and Okomoto signature schemes [17].

2.2 eIDAS pseudonymous signatures

Our proposal is basically the encryption pendant to pseudonymous signatures. Therefore we briefly show the construction pseudonymous signatures from the eIDAS specification (for details, see [13],ch. 3.7). Please note that we have adapted the variable names from [13] in order to match the notation used our paper.

Pseudonymous signatures need a group manager $\mathcal{GM}$ which has long term private/public keys $Pk_{\mathcal{D}} := sk_{\mathcal{D}} * G$ and $Pk_{\mathcal{GM}} := sk_{\mathcal{GM}} * G$. For each user in the group, the group manager calculates a pair of private keys sk_1 and sk_2. The calculation of the keys for each user is done such that $sk_{\mathcal{D}} = sk_1 + sk_{\mathcal{GM}} * sk_2$. This ensures that all users of the group share the same public key $Pk_{\mathcal{D}}$ and are thus indistinguishable. On the other hand, each user has a different private key pair sk_1, sk_2, which implies that each user will calculate different pseudonyms. The pseudonymous signature is then calculated as

eIDAS pseudonymous signature $(sk_1, sk_2, J, Pk_{\mathcal{D}}, Pk_{\mathcal{GM}}, m)$

1: $k_1; k_2 \leftarrow\$ \{0,1\}^{\kappa}$
2: $R = k_1 * G + k_2 * Pk_{\mathcal{GM}}$
3: $A_1 = k_1 * J; A_2 = k_2 * J$
4: $K_1 = sk_1 * J; K_2 = sk_2 * J$
5: $c = h(R, A_1, A_2, J, K_1, K_2, m)$
6: $s_1 = k_1 - sk_1 * c; s_2 = k_2 - sk_2 * c$
7: **return** (c, s_1, s_2)

eIDAS signature verification $(c, s_1, s_2, K_1, K_2, Pk_{\mathcal{D}}, Pk_{\mathcal{GM}}, m)$

1: $R = s_1 * G + s_2 * Pk_{GM} + c * Pk_{\mathcal{D}}$
2: $A_1 = c * K_1 + s_1 * J$
3: $A_2 = c * K_2 + s_2 * J$
4: $c' = h(R, A_1, A_2, J, K_1, K_2, m)$
5: **return** $c \stackrel{?}{=} c'$

Note that the eIDAS protocol generates two pseudonyms K_1 and K_2 for the same base point J.

2.3 Trusted Platform Module

A Trusted Platform Module (TPM) is a securely protected and tamper-resistant cryptographic co-processor [1]. It serves as a secure repository for cryptographic keys and other sensitive data intimately tied to the physical host device. Specified by the Trusted Computing Group (TCG) for general-purpose computer systems [15] The TPM encompasses cryptographic primitives for public-key cryptography, key generation, cryptographic hashing, and random-number generation.

It delivers a suite of security services, including secure boot, remote attestation, and encryption key management. A secure boot ensures that the system is initiated by a trusted source, the Root of Trust (RoT), preventing unauthorized software from being executed. Remote attestation enables a trusted entity to verify the system's integrity, ensuring it remains untampered. Key management facilitates the generation and secure storage of encryption and signing keys protected by the TPM.

Within the realm of TPM 2.0, four prevalent implementations exist: Discrete, integrated, firmware, and software TPM. The discrete TPM, often referred to as hardware TPM, stands out for providing an exceptionally high level of security.

The TPM has two types of classifications of keys: restricted and unrestricted. Restricted keys come with usage restrictions imposed by the TPM. These restrictions may include limitations on the cryptographic operations the key can perform or the data it can access. Restricted keys are often tied to specific purposes or functions. They are used to sign or decrypt TPM states or challenges. A prominent command that uses restricted keys is, for example, TPM2_ActivateCredential. Unrestricted keys have fewer usage restrictions. They are more versatile and can be used for a broader range of cryptographic operations without as many limitations as restricted keys, i.e., intended for general use. Here, one prominent representative, for example, is the TPM2_ECDH_ZGEN.

3 RELATED WORK

As mentioned in the Background section, one direct related concept is the Anonymous Authenticated Credential Key Agreement (AACKA) protocol. In this section, we cover the information about the AACKA protocol. For the sake of completeness, we briefly discuss other Privacy Preserving Authenticated Key Exchange (PPAKE) schemes.

3.1 Anonymous Authenticated Credential Key Agreement

Schermann *et al.* [18] developed a new protocol called Anonymous Authenticated Credential Key Agreement (AACKA) to enable Anonymous Authenticated Credential based Encryption (AACE). This protocol uses pairing-friendly cryptography and can be seen as a variant of a static Diffie-Hellman protocol, but instead of an X.509 certificate, Camenish-Lysyanskaya (CL) credentials [8] are used. This credential has to be negotiated with an Issuer during the Join phase. After a successful run, the Device received valid CL-credentials. After the Join phase, the AACKA phase is performed between the Device $\mathcal{D}$ and the Service Provider $\mathcal{SP}$ to negotiate a shared secret and derive a symmetric encryption key. Further, $\mathcal{D}$ has authenticated to $\mathcal{SP}$ that the private AACKA key f belongs to a valid CL-credential. The protocol is shown below. Here, they also included a Privacy Proxy $\mathcal{PP}$. Its task is to randomize the CL-credential to increase performance. As in the protocol required $\mathcal{SP}$ has to verify the credential with a bilinear mapping function, which gives an additional overhead in the protocol flow. The AACKA protocol can also be enhanced with pseudonyms to be linkable as long as the chosen basename (bsn) remains the same. This protocol is called Pseudonymous Authenticated Credential Key Agreement (PACKA). In this protocol $\mathcal{D}$ chooses a basename. Out of this bsn the pseudonyms J, K are constructed that can be linked by the $\mathcal{SP}$.

3.2 Other PPAKE methods

Privacy-preserving authenticated key exchange (PPAKE) authenticates two parties with each other and further protects the privacy

of both parties against an MITM adversary. In the following, we provide a rough overview of three noteworthy protocols within the PPAKE family [12, 14, 16]. Fan *et al.* introduced a Privacy-Preserving Authenticated Key Agreement Protocol aimed at mobile emergency services, utilizing Smart Cards [12]. This protocol ensures user anonymity even if an adversary manages to compromise the Smart Card, protecting against Man-in-the-Middle (MITM) attacks or any adversary capable of extracting information from the Smart Card. It also offers forward secrecy and heavily depends on hash functions for security. On the other hand, Ferreira developed a Privacy-Preserving Authenticated Key Exchange specifically tailored for Constrained Devices [14], enhancing the privacy features of the Symmetric-Key Authenticated Key Exchange (SAKE) protocol [3], a two-party Authenticated Key Exchange (AKE) mechanism. Similar to Fan *et al.* protocol, Ferreira's design also guards against MITM attacks, ensuring privacy, and offers robust forward secrecy. Those two mentioned protocols do not provide cross-domain unlinkability to the communicating party. Last, Liang *et al.* [16] developed a physically secure and conditionally private authenticated key agreement scheme for Vehicular Ad Hoc Networks (VANETs). This protocol is designed to accomplish mutual authentication and key agreement between vehicles and Road Side Units (RSUs). In their publication, they combined the strengths of the Elliptic Curve Cryptosystem and physically unclonable function (PUF). It provides protection against physical impersonation, MITM, replay, known session keys, and ephemeral secret leakage (ESL) attacks. Further, it ensures conditional privacy, i.e., only the Trusted Authority (TA) can determine the actual identity of a vehicle from its pseudonym. This protocol implies that each individual pseudonym must be provided by the Trusted Authority (TA) to the RSU and vehicle. This is a contrast to our approach where the device itself is capable to compute different pseudonyms.

4 DEFINITION OF OUR PAKA PROTOCOL

This section begins with an overview of the notation and conventions used in the paper. Subsequently, security definitions, as well as cryptographic hard problems, are presented.

Our protocol consists of three roles: (1) Device $\mathcal{D}$, (2) Service Provider $\mathcal{SP}$, and (3) Group Manager $\mathcal{GM}$. Note that each $\mathcal{SP}$ belongs to a domain, which is also called a sector. In general, each domain can contain several $\mathcal{SP}$. For simplicity, we consider one $\mathcal{SP}$ per domain in our protocol.

4.1 Notation

This paper uses the following symbols and abbreviations throughout the paper (see table 1): We describe the operations on elliptic curves in additive notation.

4.2 Security definitions

Our PAKA protocol is 'secure' if it fulfills the following properties: (1) correctness, (2) confidentiality + forward-secrecy, (3) unforgeability, and (4) Cross-domain unlinkabilty+forward-privacy. PAKA protocol ensures privacy in form of cross-domain unlinkabilty of the device. Cross-domain unlinkabilty means that identities in different domains or contexts cannot be linked together.

Table 1: Symbols and abbreviations

Symbol	Description
$\mathcal{D}$	Device.
$\mathcal{SP}$	Service Provider.
$\mathcal{GM}$	Group Manager (Issuer)
$\mathcal{H}$	Host
bsn	Basename.
p, n	Prime numbers.
$\mathbb{Z}_p, \mathbb{Z}_n$	prime field of characteristic p (resp. n)
EC	elliptic curve over $\mathbb{Z}_p$
$\mathbb{G}$	Cyclic subgroup of order n of EC
G	Generator of $\mathbb{G}$
\$	Random element.
$H_{\mathbb{G}}$	Hash to $\mathbb{G}$.
K_1, K_2	Pseudonyms, i.e., point in $\mathbb{G}$ for linking.
sk_1, sk_2	Device/PAKA private keys.
U_1, U_2	Ephemeral keys.
$sk_{\mathcal{GM}}, Pk_{\mathcal{GM}}$	Secret/Public key Group Manager.
$sk_{\mathcal{D}}, Pk_{\mathcal{D}}$	Pseudonymous Secret/Public key sector
Z	Shared secret.
k	Key for encryption and decryption.
kdf	Key derivation function.
Enc	Encryption function.
Dec	Decryption function.
r, r_1, r_2	Random numbers in $\mathbb{Z}_n$.

Correctness: Correctness means that an honest Device $\mathcal{D}$ can successfully perform a key agreement with an honest Service Provider $\mathcal{SP}$, i.e., the ciphertext cannot be decrypted with any other secret key.

Confidentiality means that an evesdropper cannot decrypt any session negotiated between the Device $\mathcal{D}$ and a honest Service Provider $\mathcal{SP}$. Confidentiality and forward-secrecy in our definition are directly related.

Forward-secrecy means in our protocol that the confidentiality is ensured even if the group manager keys are revealed.

Cross-domain unlinkability is our main privacy-feature and means the following. Within a sector/domain, the user should consistently appear under the same pseudonym (linkable). However, across different domains the user has different pseudonyms. It must be ensured that, for example, Domain A should not recognize the Devices pseudonyms for Domain B. In other words, an adversary seeing two pseudonyms belonging to different domains cannot decide if both pseudonyms belong to the same Device or two different ones.

Forward-privacy means cross-domain unlinkability is maintained even if the group manager keys are reveled.

Unforgeability means that one device is not able to impersonate another device. In particular, this comprises two scenarios:

- Only honest devices shall be able to perform a successful key-agreement, i.e., the device must know a private key pair (sk_1, sk_2) which fulfill $Pk_{\mathcal{D}} = sk_1 * G + sk_2 * P_{\mathcal{GM}}$
- Any device can only use his own pseudonyms and not use other pseudonyms to impersonate another device, i.e., the

device is forced to compute his pseudonyms as $K_1 = sk_1 * J_1$ and $K_2 = sk_2 * J_2$

Note: if the secret keys of the device are revealed then cross-domain unlinkabity cannot be assured. Forward-secrecy can be easily achieved in this case by using an additional ephemeral ECDH before the PAKA phase starts. This is a standard approach which is therefore not described in this paper. We would also empathize that is good practice to use a Secure Element like a TPM for storing the secret keys sk_1 and sk_2.

4.3 Cryptographic hard problems

In this subsection, we revisit cryptographic hard problems that are relevant to our proposed protocol. One problem is the well-known *Decisional Diffie-Hellman (DDH) Problem*, which was introduced by [9]: Given $H \in \mathbb{G}$ and a, b, c are random numbers of $\mathbb{Z}_n$, distinguish $(a * H, b * H, c * H)$ from $(a * H, b * H, ab * H)$. The second one is the *Discrete Logarithm Problem* : Given $G_1, G_2 \in \mathbb{G}$, calculate $l \in \mathbb{Z}_n$ which fulfills $G_2 = l * G_1$.

5 OUR NOVEL PAKA PROTOCOL

In this section, we propose our protocol that enables Pseudonymous Authenticated Key Agreement without the use of pairing-friendly cryptography and CL-credentials. This allows anonymous authenticated encryption and introduces anonymity to the device in case of cross-domain unlinkability. It starts with a setup phase, the PAKA main phase. Then, we present the integration of a TPM with the PAKA protocol, followed by the integration of PAKA.

5.1 Setup

In this phase, a trusted Group Manager, sometimes called an Issuer, has to set global domain and user-specific parameters. We assume that that setup phase is executed in a secured and trusted environment.

The scheme employs a prime order q group denoted as $\mathbb{G}$, where ensuring the hardness of the DDH Problem for $\mathbb{G}$ is crucial. Additionally, $\mathbb{G}$ must be specified to indicate the presence of a generator $g \in \mathbb{G}$. The Group Manager $\mathcal{GM}$ is responsible for generating its system keys during the setup process:

- $\mathcal{GM}$ secret keys: $sk_{\mathcal{D}}, sk_{\mathcal{GM}} \in \mathbb{Z}_q$
- $\mathcal{GM}$ public keys: $Pk_{\mathcal{D}}, Pk_{\mathcal{GM}} \in \mathbb{G}$
- user secret keys: $sk_1, sk_2 \in \mathbb{Z}_q$ in a way that $sk_{\mathcal{D}} = sk_1 + sk_2 * sk_{\mathcal{GM}}$

Note: $Pk_{\mathcal{D}} = sk_1 * G + sk_2 * Pk_{\mathcal{GM}} = sk_1 * G + sk_2 * (sk_{\mathcal{GM}} * G)$ The user keys are injected in a device by the $\mathcal{GM}$ and calculated for each member:

- user secret keys: $sk_1, sk_2 \in \mathbb{Z}_q$ in a way that $sk_{\mathcal{D}} = sk_1 + sk_2 * sk_{\mathcal{GM}}$, i.e.:
  - choose $sk_2 \xrightarrow{\$} \mathbb{Z}_q$
  - calculate $sk_1 = sk_{\mathcal{D}} - sk_{\mathcal{GM}} * sk_2$.

  This is a linear equation in two variables (sk_1, sk_2). Such an equation has one degree of freedom.
- issuer public keys: $Pk_{\mathcal{D}}, Pk_{\mathcal{GM}} \in \mathbb{G}$

5.2 Pseudonymous Authenticated Key Agreement - PAKA

This subsection shows our novel protocol to enable Pseudonymous Authenticated Encryption (PAE), i.e., exchange a pseudonymous and authenticated symmetric encryption key. It acts like a fundamental building block for an encryption scheme in a similar way as ECIES protocol is constructed from a standard key agreement. This involves a pseudonymous authentication through an ephemeral/static Diffie-Hellman key agreement between Device $\mathcal{D}$ and Service Provider $\mathcal{SP}$, i.e., $\mathcal{D}$ is pseudonymous to $\mathcal{SP}$.

The protocol uses two pseudonyms, K_1 and K_2. $\mathcal{D}$ is in possession of the secret keys sk_1 and sk_2, which are calculated by the Group Manager and personalized to each device. Those keys are used to compute the pseudonyms with the help of the point J, which is calculated with a hash function and a chosen basename *bsn*. The pseudonyms K_1 and K_2 and the *bsn* are then sent to $\mathcal{SP}$. The Service Provider $\mathcal{SP}$ chooses three random values r_1, r_2, r_3 and computes two ephemeral keys U_1 and U_2 that are sent to $\mathcal{D}$ as shown in the protocol. Now, both can compute the same shared secret and derive a symmetric encryption key. Upon completion of the PACKA protocol, the ensuing attributes are as follows: (1) $\mathcal{D}$ and $\mathcal{SP}$ have calculated the same shared secret Z. Subsequently, the same symmetric encryption key k. (2) $\mathcal{D}$ has implicitly authenticated to $\mathcal{SP}$ by proving that the keys sk_1 and sk_2 belong to his pseudonym. Note, as mentioned in subsection 1, the authentication process from $\mathcal{SP}$ to $\mathcal{D}$ can utilize any standard authentication method, as there are no specific privacy requirements for $\mathcal{SP}$. The derived shared key k can be employed following the conventions of a typical static-ephemeral Diffie-Hellman protocol. Our examination of security in section 6 indicates the presence of a slight privacy concern stemming from the potential for a replay attack (U_1, U_2). Nevertheless, addressing this issue is straightforward by implementing a basic cryptographic mechanism, like applying a nonce. To transform the pseudonymous approach into an anonymous $\mathcal{D}$ can create a new random *bsn* for each protocol invocation.

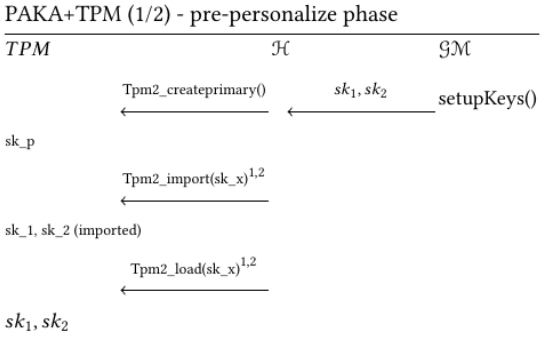
PAKA protocol

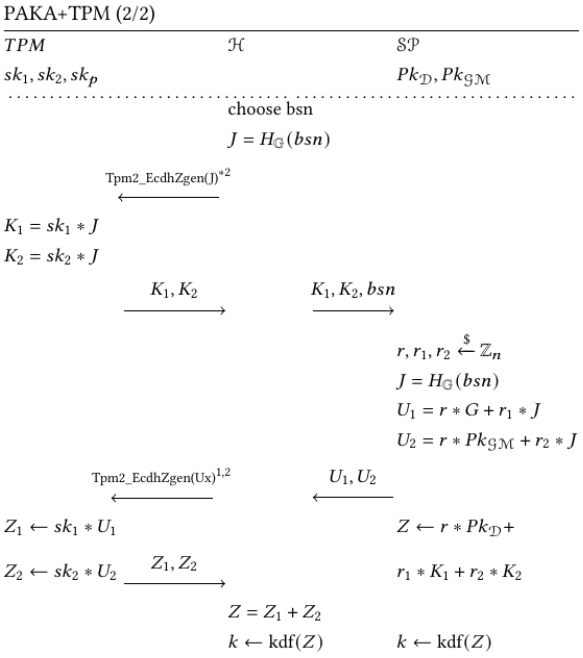
$\mathcal{D}$		$\mathcal{SP}$
$sk_1, sk_2,$		$Pk_{\mathcal{D}}, Pk_{\mathcal{GM}}$
choose bsn		
$J = H_{G1}(bsn)$		
$K_1 = sk_1 * J$		
$K_2 = sk_2 * J$		
	$\xrightarrow{K_1, K_2, bsn}$	
		$r, r_1, r_2 \xleftarrow{\$} \mathbb{Z}_n$
		$J = H_{\mathbb{G}}(bsn)$
		$U_1 = r * G_1 + r_1 * J$
		$U_2 = r * Pk_{\mathcal{GM}} + r_2 * J$
	$\xleftarrow{U_1, U_2}$	
$Z \leftarrow sk_1 * U_1 + sk_2 * U_2$		$Z \leftarrow r * Pk_{\mathcal{D}} +$
		$r_1 * K_1 + r_2 * K_2$
$k \leftarrow \mathrm{kdf}(Z)$		$k \leftarrow \mathrm{kdf}(Z)$

5.3 Integration of TPM2.0 into PAKA

In this subsection, we show how a Trusted Platform Module can be used as a secured cryptographic co-processor to perform certain cryptographic secured functions and to store the secret keys sk_1 and sk_2 securely. For this reason, we split the Device role $\mathcal{D}$ into a Host $\mathcal{H}$ and TPM. Technically, the integration can happen with two different approaches. (1) load an external key - Esys_LoadExternal command (2) import a key - Esys_import + Esys_load commands. Security wise it is not recommended to use the Esys_LoadExternal in this protocol. This is do to the fact that we must load in our protocol a public and private portion into the TPM 2.0, i.e., such a key is not allowed to be loaded into a hierarchy, except the NULL. This also implies that the NULL hierarchy is cleared for each new power cycle. Therefore, the key has to be also stored in software to be loaded again into the TPM2.0 and this withdraws the overall benefits of the TPM2.0 to make the whole architecture more secured. Nevertheless, this is not the case with the Esys_import + Esys_load process. In our protocol, we use the (2) approach.

In the pre-personalize phase (1/2), $\mathcal{H}$ must execute the Tpm2_createprimary command that the TPM2.0 creates an ECC NISTP256 AES128 CFB mode primary key (sk_p). Alternatively, a RSA 2048 can be used. Under this key, the TPM can import the two secret keys sk_1 and sk_2, received by the Group Manager. To use those keys later in the protocol, Tpm2_load for both keys has to be executed. If there is a need to make the keys permanent and persistent, the TPM2_EvictControl has to be executed for each key. After this pre-personalize phase, $\mathcal{H}$ computes a point J and uses the Tpm2_EcdhZgen command with each loaded private key (sk_1, sk_2) to calculate the pseudonyms K_1, K_2. This pseudonyms, together with the basename (bsn), are sent to $\mathcal{SP}$. $\mathcal{SP}$ computes two ephemeral keys U_1, U_2 that are transmitted back to $\mathcal{H}$. $\mathcal{H}$ calculates the shared secrets Z_1, Z_2 with the help of the TPM2.0. The final step is to add the two points together to compute the final shared secret Z that is also the same as in $\mathcal{SP}$. The addition of two points is not supported by the TPM2.0. Z can be used to derive a symmetric shared key k between both parties.





5.4 Integration of PAKA into ECIES

Within this subsection, we focus on highlighting PAKA's integration into the Elliptic Curve Integrated Encryption Scheme (ECIES) protocol to enable privacy-preserving encryption. However, it is worth noting that the PAKA protocol is adaptable to various encryption schemes, such as ElGamal. ECIES constitutes a unified hybrid encryption scheme that merges a Key Encapsulation Mechanism (KEM) with a Data Encapsulation Mechanism (DEM).

In the protocol, the Device $\mathcal{D}$ receives in the end an encrypted package or message pt from the Service Provider $\mathcal{SP}$. The protocol works as follows: First, $\mathcal{D}$ sends the pseudonyms K_1 and K_2 and the corresponding chosen base name bsn to $\mathcal{SP}$. $\mathcal{SP}$ can calculate with the help of bsn the same point J. After, $\mathcal{SP}$ computes two ephemeral keys U_1 and U_2 that have to be sent back to $\mathcal{D}$. Both sides can now calculate the same shared secret and the derived key for the verification of the Tag and the encryption/decryption process.

PAKA-ECIES protocol

$\mathcal{D}$		$\mathcal{SP}$
sk_1, sk_2		$Pk_{\mathcal{D}}, Pk_{\mathcal{GM}}$
choose bsn		
$J = H_{\mathbb{G}}(bsn)$		
$K_1 = sk_1 * J$		
$K_2 = sk_2 * J$		
	$K_1, K_2, bsn \longrightarrow$	
		$r, r_1, r_2 \xleftarrow{\$} \mathbb{Z}_n$
		$J = H_{\mathbb{G}}(bsn)$
		$U_1 = r * G + r_1 * J$
		$U_2 = r * Pk_{\mathcal{GM}} + r_2 * J$
		$Z \leftarrow r * Pk_{\mathcal{D}} + r_1 * K_1 +$
		$r_2 * K_2$
		$(k_{Mac}, k_{Enc}) \leftarrow \text{kdf}(Z)$
		$c \leftarrow \text{Enc}(pt, k_{Enc})$
		$tag \leftarrow \text{MAC}(c, k_{Mac})$
	$U_1, U_2, c, tag \longleftarrow$	
$Z \leftarrow sk_1 * U_1 + sk_2 * U_2$		
$(k_{Mac}, k_{Enc}) \leftarrow \text{kdf}(Z)$		
$tag' \leftarrow \text{MAC}(c, k_{Mac})$		
if tag' = tag		
$pt \leftarrow \text{Dec}(c, k_{Enc})$		

5.5 Design discussion

In this subsection, we highlight specific design choices that might raise questions among readers. Before we go into potential question candidates we want to clarify our design rescission.

The reader may think that the protocol could be simplified by using a standard static-ephemeral Diffie-Hellman with a group key and putting the pseudonym into the key derivation function for the session key generation. This approach has the following disadvantage: Unforgeability is no longer ensured because a dishonest device can generate an arbitrary pseudonym. Our approach enforces that the device can only use pseudonyms related to the device's private keys. For details, see the unforgability security proof in 6.1.4.

One could ask the question why $\mathcal{D}$ needs $\mathcal{GM}$ for the creation of the keys sk_1 and sk_2. It is not possible that $\mathcal{D}$ creates the keys because the keys of $\mathcal{D}$ must be calculated by using $\mathcal{GM}$'s private keys $sk_{\mathcal{D}}$ and $sk_{\mathcal{GM}}$. This ensures that all $\mathcal{D}$s get the same public key and are therefore indistinguishable.

Second, how to deal with a compromised $\mathcal{GM}$, i.e., $\mathcal{GM}$ keys are broken. In case $\mathcal{GM}$ private keys are revealed, the system is no longer secure because new valid pseudonyms can be generated without involving the group manager. This problem can be mitigated by changing the public key after having created a particular batch of sk_1 and sk_2 keys. If the group key related to a batch is revealed, then not the whole system is broken; only the part that belongs to the batch is broken. The size of the batch must be carefully chosen. It is a trade-off between the impact of a break and privacy. A small batch size means that only a few devices are affected by a break, but privacy is weak because the set of devices having the same group public key is small. On the other hand, a large batch size means that many devices are affected in case of a break, but privacy is maintained due to the large number of devices using the same public key.

Third, can $\mathcal{D}$ appear fully anonymous to a $\mathcal{SP}$. The protocol could choose a different basename *bsn* every time. However, whether this is accepted or not is part of $\mathcal{SP}$'s policy. It cannot be solved by cryptographic means alone. The Service Provider may only accept pseudonyms related to a specific basename *bsn*.

6 SECURITY ANALYSIS

Our security analysis is twofold. We start with a mathematical handcrafted proof followed by a symbolic verification done using a formal automatic tool, Proverif [6].

6.1 Mathematical Proof

In this subsection we proof the security definitions in 4.2.

6.1.1 ***Correctness.*** For correctness, one must show that if $K_1 = sk_1 * J, K_2 = sk_2 * J$ and $P_{\mathcal{D}} = sk_1 * G + sk_2 * Pk_{\mathcal{GM}}$, then the shared secret Z is the same on both sides.

- $\mathcal{D}$ derived (from protocol): $sk_1 * r * G + sk_1 * r_1 * J + sk_2 * r * Pk_{\mathcal{GM}} + sk_2 * r_2 * J$
- $\mathcal{SP}$ derived (from protocol): $r * sk_1 * G + r * Pk_{\mathcal{GM}} * sk_2 + r_1 * sk_1 * J + r_2 * sk_2 * J$

6.1.2 ***Confidentiality & forward-secrecy.*** For confidentiality, we show that the protocol is at least as secure as a static-ephemeral Diffie-Hellman with sk_1 as a private key. We show that this is even the case for a partially broken protocol, i.e. when $sk_{\mathcal{GM}}$ and $sk_{\mathcal{D}}$ have been extracted and are publicly available.

For the private key pair (sk_1, sk_2) of the device $\mathcal{D}$ and the shared secret Z, sk_2 can be calculated as $(sk_{\mathcal{D}} - sk_1)/sk_{\mathcal{GM}}$. The calculation of the Device $\mathcal{D}$ can then be written as

$$sk_1 * U_1 + \frac{sk_{\mathcal{D}} - sk_1}{sk_{\mathcal{GM}}} * U_2 = sk_1 * (U_1 - \frac{1}{sk_{\mathcal{GM}}} * U_2) + \frac{sk_{\mathcal{D}}}{sk_{\mathcal{GM}}} * U_2$$

Under the assumption that the group manager keys have been revealed, the term $\frac{sk_{\mathcal{D}}}{sk_{\mathcal{GM}}} * U_2$ is a publicly calculable value. Therefore, it can be ignored for the security analysis. One can see now that the Device effectively calculates a static Diffie-Hellman with its private key sk_1 and the ephemeral point $U := U_1 - \frac{1}{sk_{\mathcal{GM}}} * U_2$.

The Service Provider gets the two pseudonyms K_1 and K_2 from the Device. However, K_2 can be calculated as

$$K_2 = \frac{sk_{\mathcal{D}} - sk_1}{sk_{\mathcal{GM}}} * J = \frac{sk_{\mathcal{D}}}{sk_{\mathcal{GM}}} * J - \frac{1}{sk_{\mathcal{GM}}} * K_1$$

Thus, K_2 can be calculated by $\mathcal{SP}$ from K_1 and therefore contributes no additional information and can be ignored. It is not difficult to verify that $\mathcal{SP}$ effectively calculates

$$(r_1 - \frac{r_2}{sk_{\mathcal{GM}}}) * K_1 + \frac{sk_{\mathcal{D}}}{sk_{\mathcal{GM}}} * U_2$$

Ignoring the second publicly calculable term as above, one can see that the protocol is equivalent to a static-ephemeral Diffie-Hellman

over the base point J with Device private key sk_1 and the Service Provider ephemeral key $(r_1 - \frac{r_2}{sk_{\mathcal{GM}}})$.

We emphasize that this security proof shows that the private key of the Device and all the negotiated session keys remain secure even in the case that the Group Manager keys have been broken.

6.1.3 ***Cross-domain unlinkability & forward-privacy.*** For unlinkability, we must show that pseudonyms (K_1, K_2) and (K_1', K_2') stemming from two different base names *base* and *base'* cannot be related. To also show forward-privacy, we continue with the assumption that the Group Manager private keys have been revealed and show that unlinkability is nevertheless maintained. As shown above, it is sufficient to consider K_1 only. Let J and J' be two points belonging to different base names and assume they are related by the unknown discrete logarithm ι, i.e., $J' = \iota * J$. In order to decide if two pseudonyms J, K and J', K' belong to the same private key sk_1 or to a different one, the attacker must be able to distinguish if in the DH quadruple $(J, sk_1 * J, \iota * J, K')$, the point K' is calculated by $sk_1 * J'$ or by some different (i.e., random) private key sk_1'. The confidentiality proof above has shown that the private key sk_1 is secure and unknown to the attacker. Also, because J and J' are calculated by Hashes, the discrete logarithm ι between those points is also unknown. However, we are in the exact same situation as the Decisional Diffie-Hellman assumption. This means that the attacker cannot distinguish if the pseudonyms belong to the same Device or to different ones.

6.1.4 ***Unforgeability.*** Unforgeability comprises the two scenarios as described in 4.2. To show unforgeability, we can no longer make the assumption that the Group Manager keys are broken. It is easy to see that with broken Group Manager keys arbitrary values sk_1, sk_2 can be calculated by everyone and therefore, the authenticity of the public key Pk_D and the pseudonyms can no longer be guaranteed. Therefore, we will assume that the Group Manager keys $sk_{\mathcal{D}}$ and $sk_{\mathcal{GM}}$ are not revealed.

It is important to emphasize that robust physical protection of the Device keys is necessary. Indeed, if two sets of Device keys sk_1, sk_2 and sk_1', sk_2' are extracted, then the Group Manager keys can be calculated by:

$$\begin{aligned} sk_{\mathcal{GM}} &= (sk_1 - sk_1')(sk_2' - sk_2)^{-1} \\ sk_{\mathcal{D}} &= sk_1 + sk_{\mathcal{GM}} sk_2 \end{aligned}$$

Please note that this extraction attack is the same as the one used in pseudonymous signatures by Bender *et al.* [5]. For the remaining part of this security proof, we assume now that the Group Manager keys have not been revealed.

First, we show that the difficulty of finding a discrete logarithm representation $Pk_D := sk_1 * G + sk_2 * Pk_{\mathcal{GM}}$ is at least as hard as the difficulty of calculating a discrete logarithm. To show that, assume an attacker has access to an oracle that calculates for arbitrary G' and G'' values sk_1, sk_2 with $G'' = sk_1 * G + sk_2 * G'$. Given any point H, we show how this oracle can used to calculate the discrete logarithm h with $h * G = H$. Choose an arbitrary integer m and send the oracle $G' := m * G$ and H. The oracle return sk_1, sk_2 with $H = sk_1 * G + sk_2 G' = sk_1 * G + sk_2 m * G$. The attacker can then calculate the discrete logarithm $h = sk_1 + m * sk_2$.

Second, we show the unforgeability of the pseudonyms. This means we will show that the Device $\mathcal{D}$ is enforced to calculate K_1 and K_2 correctly, i.e., by using its private keys $K_1 = sk_1 * J$ and $K_2 = sk_2 * J$. We perform calculations in the logarithmic space over the point G. To simplify notation, we use the following convention. If capital X denotes a public point, the lower letter x stands for its discrete logarithm over G, i.e., the value x, which fulfills $X = x * G$. In the PAKA protocol of section 5.2, we can then write $u_1 = r + r_1 j$ and $u_2 = r sk_{\mathcal{GM}} + r_2 j$. For fixed u_1, u_2 one can rewrite r_1 and r_2 as a function depending on r, i.e.,

$$\begin{aligned} r_1(r) &:= (u_1 - r)/j \\ r_2(r) &:= (u_2 - r sk_{\mathcal{GM}})/j \end{aligned}$$

As u_1 and u_2 are independent of the chosen value r, this shows that r is information-theoretically hidden from the Device $\mathcal{D}$.

We assume that the dishonest Device $\mathcal{D}$ uses different values sk_1' and sk_2' for calculating its pseudonyms. The Service Provider $\mathcal{SP}$ calculates the shared secret as

$$\begin{aligned} z &= r sk_{\mathcal{D}} + r_1(r) k_1 + r_2(r) k_2 \\ &= r(sk_1 + sk_{\mathcal{GM}} sk_2) + \frac{(u_1 - r)}{j} j sk_1' + \frac{(u_2 - r sk_{\mathcal{GM}})}{j} j sk_2' \\ &= r\left((sk_1 + sk_{\mathcal{GM}} sk_2 - (sk_1' + sk_{\mathcal{GM}} sk_2')\right) + u_1 sk_1' + u_2 sk_2' \end{aligned}$$

If $(sk_1 + sk_{\mathcal{GM}} sk_2 - (sk_1' + sk_{\mathcal{GM}} sk_2') = 0$ then the Device $\mathcal{D}$ must have calculated a second correct representation sk_1', sk_2' for $Pk_{\mathcal{GM}}, Pk_D$. However, we have shown above that this is infeasible. Therefore, we assume that this term is non-zero. But this implies that z depends on the information-theoretically hidden value r. We conclude that $\mathcal{D}$ is only able to calculate the correct shared secret z, if it uses $sk_1' = sk_1$ and $sk_2' = sk_2$.

6.2 Formal verification proof

In this subsection, we describe the formal analysis we carried out on the proposed protocol. The analysis was performed using an automated tool, Proverif, which takes as input an abstract model of the protocol and a set of security properties that must be met and automatically returns whether the model enforces the security properties or whether attacks are possible against it. In the latter case, Proverif returns an attack trace describing a potential attack that violates the property. Below, we describe the abstract model of the Paka protocol used for the verification with Proverif.

6.2.1 Channel and Threat model. In our model, we gave the attacker the power to act on all messages exchanged by the honest entities during the protocol. All channels used for communication are in fact public and therefore potentially insecure. In particular, we modeled the attacker following the Dolev-Yao model: the attacker can read, modify, delete, and replay messages sent over public channels. Thus, to protect these messages, it is necessary to use secure cryptographic mechanisms. Both the attacker and honest entities in our model can perform cryptographic operations, but only with the keys in their possession. However, the attacker cannot perform brute force operations such as reversing a hash function or performing physical attacks.

It is important to highlight that the verification performed with Proverif is symbolic. A symbolic model assumes perfect cryptography. In other words, all cryptographic operations are assumed to be secure, and verification does not take into account implementation details or choices that could open up to possible vulnerabilities. For example, although operations on elliptic curves, such as point multiplication, are difficult to reverse, their security depends on factors such as the choice of curve or base points, which are not taken into account by Proverif in its analysis. We then assume that all choices made about the cryptographic algorithms used in the PAKA protocol are optimal.

6.2.2 Security properties. We verified using Proverif all the security properties described in Section 4.2 and mathematically proved above.

Correctness. With this property, we tested in Proverif that the protocol could reach the end without any attacker intervention and that, in doing so, the honest entities managed to compute the same symmetric session key. This property can be checked in Proverif using events.

$query\ event()$

Confidentiality. With this property, we verified the secrecy of the session key shared between the Device and the Service Provider. In Proverif, the secrecy of a term can be verified using a reachability property.

$query\ attacker()$

Unlinkability. With this property, we tested that two protocol executions initiated by the same device (i.e., with the same sk_1 and sk_2), but with different base names, bsn_1 and bsn_2, cannot be linked together. In other words, an attacker listening on the channel should not be able to understand that the two requests were sent from the same device. This property is verified in Proverif using observational equivalence.

$$equivalence$$
$$run\_protocol(sk_1, sk_2, bsn_1) \mid run\_protocol(sk_1, sk_2, bsn_2)$$
$$run\_protocol(sk_3, sk_4, bsn_1) \mid run\_protocol(sk_5, sk_6, bsn_3)$$

The query expressed above verifies whether two runs of the protocol executed by the same device (i.e., with the same sk_1 and sk_2) but with two different pseudonyms (i.e., bsn_1 and bsn_2) - first line - are indistinguishable for the attacker from two runs of the protocol executed by two different devices (i.e., one with sk_3 and sk_4, the other with sk_5 and sk_6) and two different pseudonyms (i.e., bsn_1 and bsn_3) - second line.

Unforgeability. With this property, we want to verify that the attacker, which does not know $sk_{\mathcal{GM}}$ and $_{\mathcal{D}}$, cannot forge two valid keys, sk_1 and sk_2, to participate in the protocol as a legitimate entity. Moreover, in case the attacker manages to steal one of the keys, e.g., sk_1, from an honest device, it would still not be able to compute the corresponding sk_2. This query in Proverif can be verified again using Events, Reachability, and Correspondence Assertions.

6.2.3 Verification results. Verification with Proverif showed that Correctness, Unlinkability, and Unforgeability are correctly enforced by the protocol. The Confidentiality property, however, may not be respected if the Service Provider is not properly authenticated to the device. In fact, if the authentication of $\mathcal{D}$ to $\mathcal{SP}$ is correctly enforced by the protocol, Proverif found that the vice versa is not true. In other words, the attacker could launch a MITM attack and impersonate the $\mathcal{SP}$ but can not learn the real identity (MITM-privacy is provided), and share a symmetric session key with $\mathcal{D}$. As described above in subsection 5.2, to provide authentication of the $\mathcal{SP}$ to $\mathcal{D}$, it is sufficient to simply add a signature on the U_1 and U_2 messages sent by $\mathcal{SP}$ to $\mathcal{D}$. The key to be used to perform the signature can be an asymmetric private key, whose corresponding public part can be contained in a digital certificate to be sent with the message. In this case, Proverif verified that all properties hold.

All Proverif source files used for the verification are publicly accessible at the following link, *https://github.com/netgroup-polito/verification-paka.*

7 IMPLEMENTATION

We implemented our Proof-of-Concept (PoC) on a Raspberry Pi 4 Model B Rev 1.1 (ARMv7 Processor rev 3) and 4GB RAM together with a TPM2.0 attached to the Raspberry Pi. On the software side, we used C++ as a programming language and for the cryptographic functions in $\mathcal{GM}$ and $\mathcal{SP}$ we applied the library called cryptocpp. Further, we implemented $\tilde{f}$n two ways. (1) only in software with cryptocpp (2) cryptocpp and the TPM2.0 hardware module with the tpm2-tss stack (esys layer). As shown in Table 2, the most time-consuming operation is the Esys_createPrimary. The key type also influences timing. For performance and security reasons, we suggest using an ECC NIST_P256 key with an AES128CFB mode. Further, the operation will only be executed once during the pre-setup phase. To make the key permanently persistent after a Tpm2_createprimary and a Tpm2_load the TPM2_EvictControl command must be executed.

8 CONCLUSION AND FUTURE WORK

In this paper, we have constructed a novel building block called Pseudonymous Authenticated Key Agreement (PAKA). It can be seen as an alternative to the AACKA protocol but does not need the computationally complex pairing-based cryptography. We have shown that our protocol provides cross-domain unlinkability & forward-privacy, confidentiality & forward-secrecy, and unforgeability. We have shown that this protocol is secure and privacy-preserving by giving a mathematical proof and an symbolic verification using Proverif. Finally, we proved the practicability by providing a Proof-of-Concept (PoC) implementation. The PoC is implemented in two ways. Stand-alone (SW-based)on a RaspberryPi4B+ and in the combination with a discrete Trusted Platform Module that serves as a secured cryptographic co-processor and storage of the private keys. In the end we analyzed the performance of the PoC.

Several points can be considered for future work. One point is to add a revocation mechanism in the protocol flow. Another point to consider would be integrating our proposed protocol as an additional variant of the Chip Authentication protocol 3 of the eIDAS specification [13]. Last, it would be interesting to study if this approach can be transferred to a post-quantum key establishment method.

Table 2: Performance measurements

Function	Average value[ms]
Setup keys ($\mathcal{GM}$)	7,96
Setup device keys ($\mathcal{GM}$)	6,37
Create random number ($\mathcal{SP}$)	0,05
Calculate U1 ($\mathcal{SP}$)	10,31
Calculate U2 ($\mathcal{SP}$)	10,37
Calculate Z ($\mathcal{SP}$)	16,71
Calculate J ($\mathcal{H}$)	0,11
Calculate K ($\mathcal{H}$)	2,70
Calculate Z ($\mathcal{H}$)	6,4
Esys_createPrimary (rsa2048:aes128cfb)	5529,19
Esys_createPrimary (eccNistP256:aes128cfb)	179,69
Esys_LoadExternal	9,23
Esys_Import	59,12
Esys_Load	14,03
Esys_ECDH_ZGen	79,45
Host curve points addition ($\mathcal{H}$)	0,06
Esys_FlushContext	3,18
Hkdf(SHA256)	0,21
Decryption (AES-128 CBC)	0,16
Encryption (AES-128 CBC)	0,15
MAC calculation (AES-128 CMAC)	0,12

ACKNOWLEDGMENTS

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Grant Agreement No. 871403. Furthermore, this project has partly received funding from the ECSEL Joint Undertaking, which funded the ADACORSA project under the grant agreement number 876019. ADACORSA is funded by the Austrian Federal Ministry for Climate Action, Environment, Energy, Mobility, Innovation and Technology between May 2020 and October 2023 (grant number 877585).

REFERENCES

[1] Will Arthur and David Challener. 2015. *A Practical Guide to TPM 2.0: Using the Trusted Platform Module in the New Age of Security* (1st ed.). Apress, USA.

[2] Hany Atlam, Robert Walters, and Gary Wills. 2018. Fog Computing and the Internet of Things: A Review. *Big Data and Cognitive Computing* 2 (04 2018). https://doi.org/10.3390/bdcc2020010

[3] Gildas Avoine, Sébastien Canard, and Loïc Ferreira. 2020. Symmetric-Key Authenticated Key Exchange (SAKE) with Perfect Forward Secrecy. In *Topics in Cryptology – CT-RSA 2020*, Stanislaw Jarecki (Ed.). Springer International Publishing, Cham, 199–224.

[4] Razvan Barbulescu and Sylvain Duquesne. 2018. Updating Key Size Estimations for Pairings. *Journal of Cryptology* 32, 4 (Jan. 2018), 1298–1336. https://doi.org/10.1007/s00145-018-9280-5

[5] Jens Bender, Özgür Dagdelen, Marc Fischlin, and Dennis Kügler. 2012. Domain-Specific Pseudonymous Signatures for the German Identity Card. In *Information Security - 15th International Conference, ISC 2012, Passau, Germany, September 19-21, 2012. Proceedings (Lecture Notes in Computer Science, Vol. 7483)*, Dieter Gollmann and Felix C. Freiling (Eds.). Springer, 104–119. https://doi.org/10.1007/978-3-642-33383-5_7

[6] Bruno Blanchet, Ben Smyth, Vincent Cheval, and Marc Sylvestre. 2018. ProVerif 2.00: automatic cryptographic protocol verifier, user manual and tutorial. *Version from* (2018), 05–16.

[7] Bundesamt für Sicherheit in der Informationstechnik(BSI). 2023. Kryptographische Verfahren: Empfehlungen und Schlüssellängen (BSI TR-02102-1). https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102.pdf?__blob=publicationFile&v=9. [Online; accessed 29-January-2023].

[8] Jan Camenisch and Anna Lysyanskaya. 2003. A Signature Scheme with Efficient Protocols. In *Security in Communication Networks*, Stelvio Cimato, Giuseppe Persiano, and Clemente Galdi (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 268–289.

[9] Ran Canetti and Mayank Varia. 2011. *Decisional Diffie–Hellman Problem*. Springer US, Boston, MA, 316–319. https://doi.org/10.1007/978-1-4419-5906-5_443

[10] Liqun Chen and Jiangtao Li. 2013. Flexible and Scalable Digital Signatures in TPM 2.0. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security* (Berlin, Germany) *(CCS '13)*. Association for Computing Machinery, New York, NY, USA, 37–48. https://doi.org/10.1145/2508859.2516729

[11] Liqun Chen, Dan Page, and Nigel P. Smart. 2010. On the Design and Implementation of an Efficient DAA Scheme. In *Smart Card Research and Advanced Application*, Dieter Gollmann, Jean-Louis Lanet, and Julien Iguchi-Cartigny (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 223–237.

[12] Ya-Jun Fan, Xue-Song Qiu, and Qiao-Yan Wen. 2017. A privacy-preserving authenticated key agreement protocol with smart cards for mobile emergency services. In *2017 IEEE 21st International Conference on Computer Supported Cooperative Work in Design (CSCWD)*. 227–232. https://doi.org/10.1109/CSCWD.2017.8066699

[13] Federal Office for Information Security. 2016. *Advanced Security Mechanisms for Machine Readable Travel Documents and eIDAS Token – Part 2: Protocols for electronic IDentification, Authentication and trust Services (eIDAS)*. Technical Guideline TR-03110 Part 2. Federal Office for Information Security.

[14] Loïc Ferreira. 2022. Privacy-Preserving Authenticated Key Exchange for Constrained Devices. In *Applied Cryptography and Network Security*, Giuseppe Ateniese and Daniele Venturi (Eds.). Springer International Publishing, Cham, 293–312.

[15] Trusted Computing Group. 2019. Trusted Platform Module (TPM). https://trustedcomputinggroup.org/work-groups/trusted-platform-module. Accessed: 2023-03-15.

[16] Yangfan Liang, Entao Luo, and Yining Liu. 2023. Physically Secure and Conditional-Privacy Authenticated Key Agreement for VANETs. *IEEE Transactions on Vehicular Technology* 72, 6 (2023), 7914–7925. https://doi.org/10.1109/TVT.2023.3241882

[17] Tatsuaki Okamoto. 1993. Provably Secure and Practical Identification Schemes and Corresponding Signature Schemes. In *Advances in Cryptology — CRYPTO' 92*, Ernest F. Brickell (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 31–53.

[18] R. Schermann, R. Urian, R. Toegl, H. Bock, and C. Steger. 2022. Enabling Anonymous Authenticated Encryption with a Novel Anonymous Authenticated Credential Key Agreement (AACKA). In *2022 IEEE 21st International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE Computer Society, Wuhan, China, 646–655. https://doi.org/10.1109/TrustCom56396.2022.00093

[19] C. P. Schnorr. 1990. Efficient Identification and Signatures for Smart Cards. In *Advances in Cryptology — CRYPTO' 89 Proceedings*, Gilles Brassard (Ed.). Springer New York, New York, NY, 239–252.

[20] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. 2016. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal* 3, 5 (oct 2016), 637–646. https://doi.org/10.1109/jiot.2016.2579198

[21] Stephan Wesemeyer, Christopher J.P. Newton, Helen Treharne, Liqun Chen, Ralf Sasse, and Jorden Whitefield. 2020. Formal Analysis and Implementation of a TPM 2.0-Based Direct Anonymous Attestation Scheme. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security* (Taipei, Taiwan) *(ASIA CCS '20)*. Association for Computing Machinery, New York, NY, USA, 784–798. https://doi.org/10.1145/3320269.3372197