Blind Issuance for Fast Keyed-Verification Anonymous Credentials

1st Kristián Klasovitý
Department of Telecommunication
Brno University of Technology
Brno, Czech Republic
xklaso00@vutbr.cz

2nd Sara Ricci
Department of Telecommunication
Brno University of Technology
Brno, Czech Republic
0000-0003-0842-4951

***Abstract*—Cryptographic credentials can be used to prove ownership of users' personal attributes, such as age, nationality, sex, or vaccine validity. Recently, a novel keyed-verification credential system designed for lightweight devices has been developed. In this paper, we propose an extension to this system that enhances the user's privacy by implementing blind issuance to the protocol. This allows for the creation of personalized credentials, that remain hidden from the issuer and anyone who would intercept the communication. To demonstrate the practicality of our extension, we tested the scheme on multiple devices and showed that the main algorithms are comparably fast as the original implementation.**

***Index Terms*—Anonymous Credentials, Blind Issuance, Two-party Computation, Homomorphic Encryption**

I. Introduction

Anonymous credentials allow access to internet-based services without necessarily needing to reveal all personal information. Specifically, users can anonymously prove the possession of required credentials and this proof cannot be linked to previous uses. This anonymization is not only towards outsiders but, if desired, also towards the service provider.

Depending on the services the information needed to be shown may vary. Anonymous credentials permit selecting which personal data to show depending on the request of the authorized service provider. This strategy is denoted as the data minimization principle, i.e., the personal data provided need to be "adequate, relevant, and limited to what is necessary in relation to the purposes for which they are processed". This principle is part of the General Data Protection Regulation (GDPR) [1]. A possible scenario could be the deployment of the European Union digital vaccine certificate. Here, the certificate holder does not need to show all his personal data such as name, date of birth, and place of residence when only needing to prove, for example, the validity of his vaccine.

A. Contribution

In this article, we propose a novel asymmetric anonymous attribute-based credentials scheme mainly designed for mobile devices such as smartphones and smartwatches. Our scheme enhances the privacy provided by the original Keyed-Verification Anonymous Credentials (KVAC) scheme [3] by adding blind issuance. This can be achieved by adding a Non-Interactive Zero-Knowledge Proof of Knowledge (NIZKPK) scheme in [2] in our `Issue` protocol. It is important to notice that, to our best knowledge, this is the first implementation complete with proof of knowledge of the aforementioned NIZKPK scheme.

In our proposal, the user is in possession of a secret key that is not shared with the issuer. Therefore, the issuer will be not able to impersonate any user. This property can also be used to protect the credentials from an attack during the issue phase as the real credentials can only be extracted by the user. Experimental results show the applicability of our scheme to the Internet of Things (IoT) environment.

Our proposal remains competitive, where `Show` and `Verify` algorithms that will be used regularly are almost unaffected with an increase of only 10% of the total computational cost. Therefore, the modified scheme still remains fast enough to be used in real-life IoT scenarios.

II. Preliminary

In this section, at first, we outline the used notation. Then we review the protocols on which our scheme is based, namely a non-interactive zero-knowledge proof of knowledge [2] and keyed-verification anonymous credentials [3].

From now on, we write $a \xleftarrow{\$} A$ when a is sampled uniformly at random from A. A secure hash function is denoted as $\mathcal{H} : \{0,1\}^* \rightarrow \{0,1\}^\kappa$, where κ is a security parameter. We describe the proof of knowledge protocols (PK) using the notation introduced by Camenisch and Stadler (CS) [4]. The protocol for proving the knowledge of discrete logarithm of c with respect to g is denoted as $\text{PK}\{\alpha : c = g^\alpha\}$. Dec stands for homomorphic decryption.

A. Non-Interactive Zero-Knowledge Proof of Knowledge

The NIZKPK [2] allows two parties (a signer and a issuer) to jointly compute a Boneh-Boyen signature $\sigma = g^{1/(sk_m+m)}$ of a signer's private message m and the issuer's secret key sk_m. In our case, the signer's message is his private key sk_i The NIZKPK [5] scheme is divided into two parts: `Setup` and `2-party Signature` protocol as shown in Algorithms 1 and 2. Algorithm 1 generates the system parameters needed to run the signature. In particular, this algorithm requires the generation of 2 RSA-modulo $\mathbf{n}$ and $\mathbb{n}$, where one is used to create the signature and the other as a commitment that check

the validity of the computation. Note that $\mathbf{n}$ size depends on the order of the considered elliptic curve q_{EC}. Moreover, $\mathfrak{n}$ needs to be generated in a way that neither the sender nor the issuer knows its factors. We refer to [2] for more details.

Algorithm 1 `Setup` $(\mathbf{1}^{\kappa})$

Consider q_{EC} a prime of the right order and κ a security parameter.
Generate an RSA-modulus $\mathbf{n}$ of size at least $|2^{3\kappa}q_{EC}^2|$, where $\mathbf{n} = pq$, $\phi(\mathbf{n}) = (p-1)(q-1)$, and $|p| = |q|$.
Consider $\mathbf{h} = \mathbf{n} + 1 \in \mathbb{Z}_{\mathbf{n}^2}$
Generate $\mathbf{g}$ of order $\phi(\mathbf{n})$ in $\mathbb{Z}_{\mathbf{n}^2}$
Generate another RSA-modulus $\mathfrak{n} = p_g \cdot q_g$, where p_g, q_g are big primes, $\phi(\mathfrak{n}) = (p_g - 1)(q_g - 1)$, and $|p_g| = |q_g|$.
Consider $\mathfrak{h} \xleftarrow{\$} \mathbb{Z}_{\mathfrak{n}}$ and $\mathfrak{g} \xleftarrow{\$} \langle \mathfrak{h} \rangle$.
return $par = (\mathbf{h}, \mathbf{n}, \mathbf{g}, \mathfrak{h}, \mathfrak{n}, \mathfrak{g}, q_{EC})$.

In Algorithm 2, the parties compute the signature $\sigma_i = g^{1/x}$ where $x \equiv (sk_m + sk_i) \cdot r_1 \ mod \ q_{EC}$. Therefore, this value is randomized and the value of $\sigma = g^{1/sk_m + sk_i}$ can only be restored by the user: $\sigma = \sigma_i^{r_1}$.

Algorithm 2 `2-party Signature` (par)

The issuer computes:
  $r \xleftarrow{\$} \mathbb{Z}_{\phi(\mathbf{n})}, r' \xleftarrow{\$} \mathbb{Z}_{\phi(\mathfrak{n})}$
  $e_1 = \mathbf{h}^{\mathbf{n}/2 + sk_m}\mathbf{g}^{r} \mod \mathbf{n}^2$
  $\mathfrak{c} = \mathfrak{g}^{sk_m}\mathfrak{h}^{r'} \mod \mathfrak{n}$
  $PK_m\{sk_m, r, r'\}$
The sender$_i$ checks PK_m and computes:
  $sk_i, r_1 \xleftarrow{\$} \mathbb{Z}_{q_{EC}}, r_2 \xleftarrow{\$} \{0...|2^{\kappa}q_{EC}|\}$, $\bar{r} \xleftarrow{\$} \{0...|2^{\kappa}\mathfrak{n}|\}$
  $e_2 = (e_1/\mathbf{h}^{\mathbf{n}/2})^{r_1}\mathbf{h}^{\mathbf{n}/2 + sk_i r_1 + r_2 q_{EC}}\mathbf{g}^{\bar{r}} \mod \mathbf{n}^2$
  $\mathfrak{c}' = \mathfrak{g}^{sk_i}\mathfrak{h}^{\bar{r}} \mod \mathfrak{n}$,
  $PK_i\{sk_i, r_1, r_2, \bar{r}\}$
The issuer checks PK_i and computes:
  $x = Dec(e_2) - \mathbf{n}/2$
  $\sigma_i = g^{1/x}$

B. Keyed-Verification Anonymous Credentials

KVAC [3] is an attribute-based authentication scheme that allows users to prove the ownership of their personal attributes. The scheme provides functions for issuing the credentials and for proving possession of them. In this scheme, the verifier must know the issuer's secret key in order to verify the user's attributes. The scheme consists of five main functions (`Setup`, `CredKeygen`, `Issue`, `Obtain`, `Show`, `ShowVerify`).

Algorithm 3 `Issue` $(sk = (x_0, ...x_n), (m_1, ..., m_n))$

The issuer receives attributes $(m_1, ...m_n)$.
The issuer computes:
  $\sigma = g_1^{\frac{1}{x_0 + m_1 x_1 + ... + m_n x_n}}$
  $\sigma_{x_1} = \sigma^{x_1}$, $\sigma_{x_2} = \sigma^{x_2}$, ..., $\sigma_{x_n} = \sigma^{x_n}$
The issuer sends $cred = (\sigma, \sigma_{x_1}, ..., \sigma_{x_n})$ to the user.

The `Setup` function outputs the system parameters par. The `CredKeygen` function generates the issuer private key sk and issuers parameters $ipar$. `Issue` takes as input the issuer's private key and attributes $(m_1, ...m_n)$ and outputs credential $cred$ as shown in Algorithm 3.

Algorithm 4 `Show` $(\langle m_i \rangle_{i=1}^n, \sigma, \langle \sigma_{x_i} \rangle_{i=1}^n, D))$

The verifier generates and sends $nonce \xleftarrow{\$} \mathbb{Z}_q$ to the user.
The user computes:
  $r, \rho_r, \rho_{m_{i \notin D}} \xleftarrow{\$} \mathbb{Z}_q$
  $\hat{\sigma} = \sigma^r$
  $t = \sum_{i \notin D} \sigma_{x_i}^{\rho_m \cdot r} \cdot g^{\rho_r}$
  $c = \mathcal{H}(D, \langle m_i \rangle_{i \in D}, t, \hat{\sigma}, par, ipar, nonce)$
  $s_r = \rho_r + cr$
  $\langle s_{m_i} = \rho_{m_i} - cm_i \rangle_{i \notin D}$
The user sends $proof = (\hat{\sigma}, t, s_r, \langle s_{m_i} \rangle_{i \notin D}, \langle m_i \rangle_{i \in D}, D)$ to the verifier.

The credential must then be sent to the user through a secure channel. `Obtain` lets the user check the validity of the credential and the attribute value. The `Show` Algorithm allows the user to prove ownership of his attributes with selective disclosure, meaning he can only disclose some of the attributes. With the `ShowVerify` Algorithm, the verifier can check the validity of the proof. The `Show` and `ShowVerify` algorithms are shown in Algorithm 4 and 5 respectively, note that D stands for the set of disclosed attributes.

Algorithm 5 `ShowVerify` $(\langle x_i \rangle_{i=0}^n, proof)$

The verifier checks that:
  $\hat{\sigma} \neq 1_{\mathbb{G}}$
  $c = \mathcal{H}(D, \langle m_i \rangle_{i \in D}, t, \hat{\sigma}, par, ipar, nonce)$
  $t \stackrel{?}{=} g^{s_r} \cdot \hat{\sigma}^{-c \cdot x_0 + \sum_{i \notin D}(x_i \cdot s_{m_i}) - \sum_{i \in D}(x_i \cdot m_i \cdot c)}$

III. Proposed Scheme

In this section, we propose an extension of the KVAC protocol by implementing blind issuance. This can be done with the NIZKPK 2-party computation and it makes the issued credential private to the user. Therefore, this credential cannot be misused by a malicious issuer to impersonate the user. To achieve these properties, we add the user's private key to the protocol, modifying the `Issue`, `Show`, `ShowVerify` algorithms, and utilizing the 2-party computation during issuing. Algorithms 6, 7 and 8 depict our proposed scheme with changes marked in red.

Algorithm 6 `Issue` $(sk = (x_0, ...x_n), (m_1, ...m_n), sk_i)$

The issuer receives attributes $(m_1, ...m_n)$.
The issuer computes: $d = x_0 + m_1 x_1 + ... + m_n x_n$.
Run 2-party computation to compute: $k = (d + sk_i) r_1$
The issuer computes:
  $\sigma^* = g_1^{1/k}$
  $\sigma^*_{x_1} = \sigma^{*x_1}$,$\sigma^*_{x_2} = \sigma^{*x_2}$,..., $\sigma^*_{x_n} = \sigma^{*x_n}$
The issuer sends $\sigma^*, \sigma^*_{x_1}, ..., \sigma^*_{x_n}$ to the user.
The user computes $\sigma = \sigma^{*r_1}, ..., \sigma_n = \sigma_n^{*r_1}$.

For our implementation, we chose the C/C++ programming language. The main reason for this is that C libraries tend to be faster for cryptography purposes. The existing KVAC code [6] and the partial NIZKPK implementation [5] were used as a starting point, that we then modified and extended to produce our scheme. The main libraries used are GMP for computation with big numbers in the NIZKPK and micro-ECC for the computation over an elliptic curve in KVAC.

Algorithm 7 `Show` $(\langle m_i\rangle_{i=1}^n, \sigma, \langle \sigma_{x_i}\rangle_{i=1}^n, D), sk_i)$

1: The verifier generates $nonce \xleftarrow{\$} \mathbb{Z}_q$ and sends it to the user.
2: The user computes:
3: $r, \rho_r, \rho_{m_{i\notin D}}, \rho_u \xleftarrow{\$} \mathbb{Z}_q$
4: $\hat{\sigma} = \sigma^r$
5: $t = \sum_{i\notin D} \sigma_{x_i}^{\rho_m \cdot r} \cdot g^{\rho_r} \cdot \sigma^{\rho_u \cdot r}$
6: $c = \mathcal{H}(D, \langle m_i\rangle_{i\in D}, t, \hat{\sigma}, par, ipar, nonce)$
7: $s_r = \rho_r + cr$
8: $s_u = \rho_u - csk_i$
9: $\langle s_{m_i} = \rho_{m_i} - cm_i\rangle_{i\notin D}$
10: The user sends $proof = (\hat{\sigma}, t, s_r, \langle s_{m_i}\rangle_{i\notin D}, s_u, \langle m_i\rangle_{i\in D}, D)$ to the verifier.

We had to integrate the NIZKPK algorithm to work with the KVAC implementation. Since GMP uses the `mpz_t` type and micro-ECC uses the `uECC_word` for storing numbers and points, we had to use conversion trough `uint8_t` to pass between these two types. Also, the previous implementation of NIZKPK did not include the calculations of the proofs of knowledge computed by the issuer and the user, so we had to implement those to make the NIZKPK complete.

Algorithm 8 `ShowVerify` $(\langle x_i\rangle_{i=0}^n, proof)$

1: The verifier checks that:
2: $\hat{\sigma} \neq 1_{\mathbb{G}}$
3: $c = \mathcal{H}(D, \langle m_i\rangle_{i\in D}, t, \hat{\sigma}, par, ipar, nonce)$
4: $t \stackrel{?}{=} g^{s_r} \cdot \hat{\sigma}^{-c\cdot x_0 + \sum_{i\notin D}(x_i \cdot s_{m_i}) - \sum_{i\in D}(x_i \cdot m_i \cdot c) + S_u}$

In the implementation, we used the *secp256_r1* curve, as a 256-bit elliptic curve provides a good level of security. We used the same security parameter κ as the previous NIZKPK implementation (1350) that follows the National Institute of Standards and Technology (NIST) security standard. In the KVAC we issued 10 attributes and shown 2 during the `Show` algorithm. The set of parameters used in the implementation are shown in Table I.

IV. Experimental Results

As mentioned before, we used C/C++ programming language. We tested the implementation on multiple devices and operating systems. Devices used were: 1) a PC: HP Pavilion 15 with OS: Windows 10 Home 20H2, RAM: 16 GB, CPU: Intel i5-9300H - 4 cores 2.4-4.1 Ghz; 2) a mobile phone: Xiaomi Redmi Note 8 Pro wit OS: Android 10, RAM: 6 GB, CPU: MediaTek Helio G90T 8 cores 2.05 GHz; 3) a smart watch: Galaxy Watch 4 Classic with OS: Wear OS 3, RAM: 1,5 GB, CPU: Samsung Exynos W920 - 2 cores 1.18 GHz; and 4) a Raspberry Pi 4 Model B with OS: Raspbian, RAM: 2 GB, CPU: ARM Cortex-A72 - 4 cores 1.5 GHz.

TABLE I
Parameters used in the implementation.

Parameter	Bitsize	Value
κ (security parameter)	32	1350
q_{EC}	256	0xffffffff00000000ffffffffffffffffbce6faada7179e84f3b9cac2fc632551
$\mathfrak{n}$,$\mathfrak{u}$ (RSA moduli)	4616	random
g	9232	random
h	4616	n+1
$\mathfrak{h}$,$\mathfrak{g}$	4616	random

Since the GMP library is built mainly for Unix systems, the library did not perform as expected on Windows PC, so we also tested the implementation on the same HP Pavilion 15 laptop in a virtualized environment running Ubuntu 64-bit Linux (VMWare, RAM: 8 GB, CPU: 4 cores). While the virtualized machine will not use the hardware of the laptop to its full potential, it still gives us some results for comparison.

Therefore, we conducted benchmarks on all the mentioned devices. The times shown in Table II are averages of 10 measures each. We grouped some of the parts of the protocol together to make our table easier to understand. In Table II, `Issuer 2-party` phase consists of computation of e_1, proof of knowledge PK_M, checking user's proof of knowledge, and decrypting of e_2, i.e., Lines 1-5,11, and 12 of Algorithm 2. `User 2-party` phase consists of times of checking issuer's proof of knowledge, computation of e_2 and creating the user's proof of knowledge PK_i, i.e., Lines 6-10 of Algorithm 2. We also compare the times of the modified KVAC with the original version to see if our changes in the `Show` and `ShowVerify` algorithms did affect the protocol's runtime. We ran the whole scheme on all devices but practically the issuer part will probably not be run on mobile devices.

Looking at the times in Table II, the `Setup` computation is the heaviest part of the scheme. Specifically, the `Setup` needs to generate the prime factors for the two RSA groups and this computation takes about 95% of the total `Setup` time. Since the GMP library only provides us with a function `mpz_nextprime` which finds the next prime after a number in parameter, we have to first generate a random number of desired bit length and then use this function. Because we are looking for prime numbers of around 2300 bits, this can take a long time. Note that the time of the `Setup` varies considerably depending on how long the search for prime numbers takes. Nevertheless, the `Setup` needs to be run only once, can be generated beforehand, and, therefore, it will not affect the time of the `Issue` algorithm in a real scenario.

Figure 1 depicts the consumption in percentage taken by the individual computations in the 2-party protocol on each side. Note that for the user the PK_M in the graph stands

TABLE II
BENCHMARKS OF THE SCHEME (TIMES IN MS).

	PC-Win 10	VM-Ubuntu	Raspbery Pi	Android Phone	Smartwatch
Our Proposal					
2-Party setup	8 512	3 063	37 089	10 697	47 359
Issuer 2-party	4 457	1 714	13 907	4 636	22 076
User 2-Party	4 414	1 833	13 787	4 625	23 076
Issue total	17 397	6 674	64 873	20 029	92 726
Issue without 2-party	14	64	90	71	215
Show	8	27	43	37	128
Verify	1,5	4	9	7	24
KVAC [3]					
Issue	7	30	39	35	132
Show	8	25	38	34	113
Verify	1,4	4	9	7	24

for checking the validity of the issuer's proof, same in PK_i for the issuer. The most time-consuming operation on both sides is computing and checking the PK_i, which is logical as it consists of the bigger number of modular exponentiations, which is a heavy operation in a big modulus.

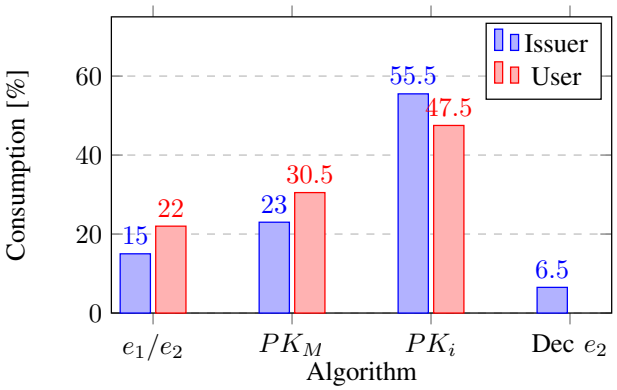

Fig. 1. Consumption used by the functions in the 2-party computation.

When we look at the total time of the `Issue` algorithm and exclude the 2-party computation we still get around twice the time than the original `Issue` algorithm, this is because there is no r_1 if we do not run the 2-party computation. This means that in our case for 10 issued attributes we have to do 10 multiplications on the elliptic curve. Note that our `Show` algorithm is only about 10% slower than the original while the `ShowVerify` takes almost the same amount of time as the original one.

As mentioned before the GMP library is built mainly for Unix systems and does not perform well on Windows. That is why the NIZKPK algorithm is so slow on the Windows PC while running almost three times faster on a Ubuntu virtual machine on the same PC, but the computations in KVAC on the elliptic curve are three times faster on the PC itself since we do not use GMP here and micro-ECC runs as good on Windows as on Unix. Therefore if we would run this algorithm on a similar PC running a Linux distribution (not virtualized), we could get times around 2.5 s for the issue algorithm. Raspberry Pi is really slow in the NIZKPK part of the scheme since this part is really memory intensive and the Raspberry Pi only has 2 GB of RAM, that is probably why the mobile phone is faster in this part while achieving similar times in the KVAC algorithms. But the Raspberry Pi could be easily used as a verifier device since the `ShowVerify` algorithm is very fast.

While true, the issue algorithm is slow, we have to consider that it only has to run once for each user, after that only the `Show` and `ShowVerify` algorithms will be run and those are almost as fast as the original, so they still should be suitable for real-life applications.

V. CONCLUSION

Anonymous credentials allow the selection of which personal data to show depending on the request of the authorized service provider. In this article, we extended the KVAC anonymous credentials scheme and show how blind issuance can be implemented into it. Moreover, to our best knowledge, we produce the first implementation of the NIZKPK scheme. While the setup algorithm of our scheme is rather slow, it only has to be run once for each user. The other algorithms of the protocol that will be used regularly are at most 10% slower than the original implementation.

REFERENCES

[1] EU General Data Protection Regulation GDPR, "Art. 5 GDPR - Principles relating to processing of personal data," GDPR.eu, November 14, 2018. Available at: https://gdpr.eu/article-5-how-to-process-personal-data/

[2] M. Belenkiy, J. Camenisch, M. Chase, M. Kohlweiss, A. Lysyanskaya and H. Shacham, "Randomizable proofs and delegatable anonymous credentials," In Annual International Cryptology Conference (pp. 108-125). Springer, Berlin, Heidelberg.

[3] J. Camenisch, M. Drijvers, P. Dzurenda, J. Hajny, "Fast keyed-verification anonymous credentials on standard smart cards," In ICT Systems Security and Privacy Protection: 34th IFIP TC 11 International Conference, SEC 2019, Lisbon, Portugal, June 25-27, 2019, Proceedings 34 2019 (pp. 286-298). Springer International Publishing.

[4] Camenisch, J., Stadler, M.: Efficient group signature schemes for large groups. In Kaliski, B., editor, Advances in Cryptology - CRYPTO '97, volume 1294 of Lecture Notes in Computer Science, pages 410–424. Springer Berlin / Heidelberg (1997).

[5] M. Sečkár, S. Ricci, "Secure Two-Party Computation for weak Boneh-Boyen Signature", Proceedings I of the 28th Conference STUDENT EEICT 2022 General Papers. 1. Brno: Brno University of Technology, 2022. s. 145-148. ISBN: 978-80-214-6029-4.

[6] T. Cvrček, "Cryptography on Arduino platform", Bachelor's thesis. Brno: Brno University of Technology, 2020.