Computer Networks 242 (2024) 110259

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

A privacy-preserving key transmission protocol to distribute QRNG keys using zk-SNARKs

David Soler [a,*], Carlos Dafonte [a], Manuel Fernández-Veiga [b], Ana Fernández Vilas [b], Francisco J. Nóvoa [a]

[a] CITIC, Universidade da Coruña, A Coruña, Spain
[b] atlanTTic, Universidade de Vigo, Vigo, Spain

ARTICLE INFO

Keywords:
QRNG
Key transmission
Zk-SNARK
Communication protocol
Privacy-preserving authentication
NFC

ABSTRACT

High-entropy random numbers are an essential part of cryptography, and Quantum Random Number Generators (QRNG) are an emergent technology that can provide high-quality keys for cryptographic algorithms but unfortunately are currently difficult to access. Existing Entropy-as-a-Service solutions require users to trust the central authority distributing the key material, which is not desirable in a high-privacy environment. In this paper, we present a novel key transmission protocol that allows users to obtain cryptographic material generated by a QRNG in such a way that the server is unable to identify which user is receiving each key. This is achieved with the inclusion of Zero Knowledge Succinct Non-interactive Arguments of Knowledge (zk-SNARK), a cryptographic primitive that allow users to prove knowledge of some value without needing to reveal it. The security analysis of the protocol proves that it satisfies the properties of Anonymity, Unforgeability and Confidentiality, as defined in this document. We also provide an implementation of the protocol demonstrating its functionality and performance, using NFC as the transmission channel for the QRNG key.

1. Introduction

Random numbers have always been a staple in cryptography. Every cryptographic algorithm, from symmetric encryption like AES to asymmetric encryption like RSA, requires random numbers as key material or initialisation parameters. The most common sources of randomness for cryptographic purposes are Pseudo-Random Number Generators (PRNG), which are deterministic processes that are initialised from a specific seed. Furthermore, PRNGs can contain vulnerabilities in their implementations [1,2] that make them vulnerable to attacks. Thus, PRNGs usually produce low-entropy key material, which can compromise the entire cryptographic scheme: no matter how theoretically secure an encryption algorithm is, it could be easily cracked if an attacker manages to obtain the secret key by exploiting its insecure generation. This is not only a theoretical issue: low-entropy key generation has been used as an entrypoint to break confidentiality in well-known cryptosystems [3–5].

Quantum Random Number Generators (QRNG) [6–8] are a different type of Random Number Generators that make use of the intrinsically unpredictable behaviour of quantum particles as their source of entropy. Because of this, the key material they generate is of higher quality than that generated by a PRNG, and its use could enhance the security of current cryptographic algorithms. Unfortunately, the technology is not yet fully developed and QRNGs are an expensive asset, so these devices cannot be widely distributed and end users have very limited access to them. In a context that may require a high level of privacy, such as private communications, users will benefit from having access to cryptographic keys with high entropy, such that it is harder for attackers to guess the key they have used. Furthermore, providing easy access to this key material could encourage users to take their privacy seriously and improve their overall communication security.

The objective of this work is to provide a method for end users to obtain higher-quality cryptographic material by defining a key transmission protocol in which a server with access to the QRNG distributes random keys in a secure manner to users that request them. As part of the process, the server will require clients to authenticate themselves before providing the QRNG key material.

However, existing Entropy-as-a-Service solutions [9] require users to trust its key distribution server, since this server can trace which keys it has provided to each user. A malicious or compromised server could use the keys it has transmitted to decrypt users' communications or even impersonate them. Indeed, the requirement to trust the server is a common concern in key distribution protocols. This work aims to provide a solution to that problem, designing a key transmission

* Corresponding author.
E-mail address: david.soler@udc.es (D. Soler).

https://doi.org/10.1016/j.comnet.2024.110259
Received 16 October 2023; Received in revised form 29 January 2024; Accepted 16 February 2024
Available online 21 February 2024

protocol that seeks a balance between requiring authentication (i.e., requesting private information from users so that access is restricted to those who can prove their identity) and protecting the users' privacy. This requirement is achieved with the inclusion of Zero Knowledge Succinct Non-interactive Arguments of Knowledge (zk-SNARK) as proofs of authentication, a cryptographic primitive that allows users to prove that they know some value without needing to reveal it, such that any verifier could be convinced of that fact.

Our protocol is divided in two main phases: in the first phase, a user performs authentication and in the second phase the user requests the QRNG key by presenting a zk-SNARK proof. The inclusion of zk-SNARKs allows us to clearly decouple the two phases, such that presenting the zk-SNARK proof in the second phase does not reveal any information (not even to the server) about the credentials used in the first phase. Thus, users can prove that they have previously authenticated themselves, in such a way that the server could not distinguish the user from any other user that has ever performed authentication. This prevents the key distribution server from associating a key to a specific user, which would allow the server to decrypt any message that uses this key. Our protocol also provides protection against replay attacks: if an attacker tries to reuse another user's zk-SNARK proof, the server will detect the attempt and abort the execution.

Since random numbers are required in all widely used cryptographic algorithms, the QRNG-generated values distributed by the protocol could be useful in a wide range of applications. For example, scenarios where users need to establish communication channels with a high level of confidentiality could employ the random numbers as high-quality key material for symmetric encryption. Another application is in multifactor authentication: possession of a random number generated by this protocol also proves that its owner has successfully performed authentication, therefore a server could require users to execute this protocol and present the QRNG-generated value to provide access.

In summary, our work presents the following contributions:

1. A Key Transmission Protocol for the distribution of QRNG keys. The protocol requires two distinct interactions between clients and server: one for authentication and another for receiving the QRNG key.
2. A privacy-preserving method of authentication for the key Transmission Protocol. The authentication is defined in such a way that it is impossible for the server to track who is receiving each key, thus preventing the server from spying on clients' communications that use these keys. This allows users to execute the protocol even if they do not trust the server generating the keys. The protocol meets the security requirements of Anonymity, Unforgeability and Confidentiality, as defined in this document. We remark that the Unforgeability property also prevents replay attacks, in which an attacker tries to present another user's proof as its own.
3. An implementation of the proposed Key Transmission Protocol. The implementation simulates a real use case, in which the server is located in some specific location users have to attend to. Part of the communication is performed through NFC and the key is stored in the users' mobile phone. Two different zk-SNARK libraries are employed for proof generation and validation, and the server uses two QRNGs for key generation. The time complexity of the protocol is estimated, measuring the proof generation time for different parameters, the NFC transmission time and the key generation rate for both QRNGs.

The rest of the document is organised as follows: Section 2 will review related works and compare them with our contribution. Section 3 will introduce the required technical background to the reader. Section 4 explains in detail all steps involved in the execution of the protocol, as well as the entities that participate in it. Section 5 discusses the security properties of the protocol. Section 6 analyses the proof-of-concept implementation of all entities involved in the protocol, and in Section 7 the performance of the implemented applications is tested. Finally, Section 8 will conclude this document and discuss future improvements.

2. Related work

Many of the recent improvements in sources of entropy have focused on Distributed Randomness Beacons (DRB) [10,11]. In these schemes, a group of users can generate a public random value that can be verified by all participants and other parties, ensuring that no user could have introduced bias in the calculation. While verifiable public randomness is useful in multiple scenarios, there is still the need for sources of entropy that distribute the generated random numbers in a private manner.

There exist multiple services that provide high entropy randomness on-demand, such as *random.org* [12]. NIST has designed an architecture for the generation and distribution of random numbers [9], which requires clients to present an authentication token. In these schemes the server knows which client is receiving each random values, and thus any cryptographic material that is generated through them. This represents a privacy concern for clients, which may not trust the server that provides the source of entropy.

The goal of authentication usually comes into conflict with the privacy of the users: the authentication server may require more information than the client is willing to give. Zero Knowledge Proofs are a family of methods by which a prover can prove to a verifier that he knows some private information without needing to reveal it. Their use in authentication protocols allow users to prove their identity without revealing sensitive information about themselves. In Non-interactive variants of Zero Knowledge Proofs, the prover and the verifier do not need to directly establish communication: instead, the prover publishes a proof that can be verified later by any other entity.

Non-interactive Zero-Knowledge authentication [13] is achieved in [14] by requiring authorised nodes to solve a hard problem (in this case, graph isomorphism), which users from outside the network would not be able to solve. Likewise, [15] also uses graph isomorphism as authentication mechanism, using a Merkle Tree as a commitment to turn the authentication into non-interactive. Both of these protocols assume a list of clients capable of solving this hard problem, making it difficult to add new authorised clients to the system.

The authors of [16] define a scheme in which users belonging to certain trust domains can authenticate each other and establish communication sessions. A method for registering new users is defined, but the defence against replay attacks is weak, since it uses timestamps: it does not protect against replays that happen in short span of time.

The scheme defined in [17] allows educational institutions to submit a learner's diploma into a public blockchain. A custom zero-knowledge proof is employed in multiple occasions, such as protecting the learner's identity or proving the grades of the learner. The protocol does not provide any direct protection against replay attacks, but replaying a proof is not sufficient to impersonate a legitimate learner: an attacker would also need to know that learner's secret key.

Among NIZK algorithms, zk-SNARKs are a popular alternative that can also be used in authentication protocols. The authors of [18] propose an authentication protocol for healthcare environments in which a set of pre-registered clients authenticate themselves with a zk-SNARK validated with a Smart Contract with the objective of establishing a secure communication channel. In [19], a certificate authority is inserted into the scheme to validate users' parameters, which they can later provide proof of. However, it presents no defence against attackers that could intercept another user's proof and later present it as their own. In [20], the focus is defining a system that can provide Linkability and Traceability to identify malicious users, while protecting the privacy of honest users. Since the authors orient their work to a voting application, the list of authorised users is predefined, and cannot be changed.

Table 1
Comparison between other works that use non-interactive zero-knowledge authentication and our proposed protocol.

Scheme	Purpose	Authentication method	Client registration	External infrastructure	Replay protection
[14]	Authentication	Isomorphism	No	No	No
[15]	Authentication	Isomorphism	No	No	No
[16]	Key agreement	Discrete logarithm	Yes	No	Timestamp
[17]	E-learning records	Zero knowledge proof	Yes	Yes (Blockchain)	Secret key
[20]	Voting	zk-SNARK	No	Yes (Blockchain)	Yes
[19]	Identity management	zk-SNARK	Yes	Yes (Blockchain)	No
[18]	Key agreement	zk-SNARK	No	Yes (Ethereum)	Yes
Ours	Key transmission	zk-SNARK/Merkle Tree	Yes	No	Yes

The concept of Commitments and Nullifiers, which are extensively used in this work, are introduced in [21]. As in the proposed protocol, in ZCash users can create a Commitment when they earn the right to perform an action and they must publish a Nullifier to consume it, accompanied by a zk-SNARK verifying the relationship between these hashes.

Table 1 compares the protocol we propose with the works mentioned in this Section. As shown, our protocol is capable of introducing new users dynamically into the system, does not require external infrastructure outside the control of the scheme, and provides protection against replay attacks.

3. Background

To achieve our privacy-preserving Key Transmission Protocol, we will need to decouple the Authentication and Key Request steps, such that the server will not be able to link clients performing the latter with the credentials they used in the former. To that end, we employ the cryptographic primitive zk-SNARK, which is formally defined in Section 3.1.

We use zk-SNARKs in this work to prove membership to a set of valid users. To speed up zk-SNARK proof generation, the aforementioned set is structured as a Merkle Tree, in which proving membership only takes $log_2(N)$ steps where N is the total number of users. Finally, we employ assymetric cryptography to securely transmit the QRNG key to the user in the last step of the protocol.

3.1. zk-SNARKs

Zero Knowledge Succinct Non-Interactive Arguments of Knowledge (zk-SNARKs) are a subset of non-interactive zero-knowledge proofs. A zk-SNARK is built from a relation R between a public Statement x and a private Witness w. If the relation between w and x holds, then $(x,w) \in R$. A security parameter λ can be derived from the description of R. Formally, a zk-SNARK scheme Π for a relation R is composed of the triplet $(\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify})$, defined as follows:

- $\mathsf{Setup}(R) \rightarrow (pk, vk)$: Takes a relation and outputs the Common Reference Key, which is divided into the Proving Key pk and the Verification Key vk.
- $\mathsf{Prove}(pk, x, w) \rightarrow \pi$: From pk, a witness w and a statement x, a proof π is generated.
- $\mathsf{Verify}(vk, x, \pi) \rightarrow Accept/Reject$: From vk, a proof π and the corresponding statement x, outputs $Accept$ or $Reject$. In the context of this work, a *valid proof* refers to any (x_v, π_v) such that $\mathsf{Verify}(vk, x_v, \pi_v) = Accept$.

A zk-SNARK scheme must also satisfy the following requirements [22]:

- Completeness. The probability

$$\Pr\left[\begin{array}{c} (pk, vk) \leftarrow \mathsf{Gen}(R) \\ \pi \leftarrow \mathsf{Prove}(pk, x, w) \\ (x,w) \in R \end{array} : \mathsf{Verify}(vk, x, \pi) = Reject\right]$$

is negligible. Intuitively, this means that an honest prover is able to convince a verifier that $(x,w) \in R$.

- Knowledge Soundness. For every efficient adversary A, there exists an efficient extractor Ext_A with access to the internal state of A such that the probability

$$\Pr\left[\begin{array}{l} ((pk, vk), aux) \leftarrow \mathsf{Gen}(R) \\ (x, \pi) \leftarrow \mathsf{A}(R, aux, (pk, vk)) \\ w \leftarrow Ext_A(R, aux, (pk, vk)) \end{array} : \begin{array}{c} (x,w) \notin R \\ \wedge \mathsf{Verify}(vk, x, \pi) = Accept \end{array}\right]$$

is negligible, where aux is an auxiliary input produced by Gen. Intuitively, this means that dishonest provers could not generate a valid proof if they do not know w.

- Zero-knowledge. For every adversary A acting as a black box and $(x,w) \in R$, there exists a simulator $\mathsf{Sim}((pk, vk), aux, x)$ such that the following equality holds:

$$\Pr\left[\begin{array}{c} ((pk, vk), aux) \leftarrow \mathsf{Gen}(R) \\ \pi \leftarrow \mathsf{Prove}(pk, x, w) \end{array} : \mathsf{A}((pk, vk), aux, x, \pi) = 1\right]$$

$$\approx$$

$$\Pr\left[\begin{array}{c} ((pk, vk), aux) \leftarrow \mathsf{Gen}(R) \\ \pi \leftarrow \mathsf{Sim}((pk, vk), aux, x) \end{array} : \mathsf{A}((pk, vk), aux, x, \pi) = 1\right]$$

where aux is an auxiliary input produced by Gen. Intuitively, this means that an attacker cannot find out anything about a witness w from a proof π, a statement x and a key pair (pk, vk).

- Weak Simulation-Extractability. For every efficient adversary A with access to an oracle O, there exists an efficient extractor Ext_A with access to the internal state of A such that the probability

$$\Pr\left[\begin{array}{c} ((pk, vk), aux) \leftarrow \mathsf{Gen}(R) \\ (x, \pi) \leftarrow A^{O(aux)}(R, aux, (pk, vk)) \\ w \leftarrow Ext_A(R, aux, (pk, vk)) \end{array} : \begin{array}{c} x \notin Q \\ \wedge (x,w) \notin R \\ \wedge \mathsf{Verify}(vk, x, \pi) = Accept \end{array}\right]$$

is negligible, where aux is an auxiliary input produced by Gen and Q is the list of x_i requested to the oracle. This means that A cannot generate a valid proof for a statement that was not obtained from the oracle. Note that there is a stronger notion of Simulation-Extractability that requires both the statement x and the proof π not to be included in Q. However, for this work we only require the weak version of Simulation-Extractability.

- Succinctness. The proof size is polynomially bounded by λ. The execution time of Verify is bounded linearly by the size of x and polynomially by λ.

In practice, zk-SNARK schemes can be built by different cryptographic techniques. The most popular designs, including Groth16 [23] (which we will use in this work) involves the transformation of a computation into an arithmetic circuit. The performance of a zk-SNARK scheme, specially during setup and proof generation, heavily depends on the number of constraints of this circuit [24], which are determined by the computation that is to be verified.

3.2. Merkle Tree

Merkle Trees [25] are a data structure in the form of a binary tree. Each element is identified as $h_{i,j}$, where i is the level of the tree and j the position inside said level. A Merkle Tree of depth n contains at most $l = 2^n$ leaf nodes $h_{n,j}$ containing a hash, and intermediate nodes take the value $h_{k,j} = H(h_{k+1,m} \parallel h_{k+1,m+1})$, where $\parallel$ is the concatenation operator, H is a hash function and $m = 2j$. The node $h_{0,0}$ is called *root*, and its value is influenced by every other node in the Tree.

The main advantage of Merkle Trees is that checking if an element e exists inside the list of leaf nodes is only $\mathcal{O}(\log l)$, while the same operation for a list would be $\mathcal{O}(l)$. This is achieved by providing a validation list $\{h_{n-1,k_{n-1}}, h_{n-2,k_{n-2}}, \dots, h_{1,k_1}\}$ and the root $h_{0,0}$ of the Merkle Tree. If it is possible to reconstruct the provided root from $H(e)$ and the validation list, then e is proven to be inside the Merkle Tree. This method requires the hash function H used to construct the Merkle Tree to be collision-resistant [26]: otherwise, it could be possible for malicious users to prove that a fake element e_f is inside the Merkle Tree when it is not.

For the scope of this work, the following operations are defined for Merkle Trees:

1. $\mathsf{EmptyTree}(n) \rightarrow T$: Initialises an empty Merkle Tree of depth n, with all of its leaf nodes set to $H(0)$.
2. $\mathsf{AddLeaf}(T, h) \rightarrow T'$: Takes a Merkle Tree T and a hash h and returns a modified tree, in which the leftmost $H(0)$ is substituted by h. The intermediate and root nodes are also updated accordingly.
3. $\mathsf{GetRoot}(T) \rightarrow R$: Takes a Merkle Tree T and returns the root $h_{0,0}$.
4. $\mathsf{GetIndexOf}(T, h) \rightarrow index$: Takes a Merkle Tree T and a hash h and returns i such that $h_{n,i} = h$ for some $h_{n,i}$ in the Tree.
5. $\mathsf{ValidationList}(T, i) \rightarrow Val = \{h_{n-1,k_{n-1}}, h_{n-2,k_{n-2}}, \dots, h_{1,k_1}\}$: Takes a Merkle Tree T and an index i and returns the validation list required to validate the leaf $h_{n,i}$.
6. $\mathsf{IsLeafOfTree}(h, i, Val, R) \rightarrow true/false$: Takes a hash h, its index i, a validation list $Val = \{h_{n-1,k_{n-1}}, h_{n-2,k_{n-2}}, \dots, h_{1,k_1}\}$ and a root R and outputs $true$ if h is the leaf with index i of the tree with root $= R$, and $false$ otherwise.

3.3. Key Encapsulation Mechanism

A Key Encapsulation Mechanism (KEM) [27,28] is a method for distributing key material between two parties employing asymmetric cryptography. The sender uses the receiver's public key to encrypt some input key material such that only the receiver can decrypt it. Both parties must agree on the asymmetric encryption algorithm that will be used for the encapsulation. In Hybrid Public Key Encryption [29], the key material is later passed through a Key Derivation Function to obtain a symmetric key.

A KEM is composed by the following operations:

- $\mathsf{KeyGen}(\lambda) \rightarrow (pk, sk)$: Takes a security parameter λ and generates a symmetric key pair.
- $\mathsf{Encap}(ikm, pk) \rightarrow c$: Encrypts some input key material ikm with a public key pk.
- $\mathsf{Decap}(c, sk) \rightarrow km$: Decrypts a ciphertext with a secret key sk to obtain the key material.

4. Privacy-preserving key transmission protocol

4.1. Outline

The protocol involves two different interactions between Users and the Server: the Authentication and the Key Request. The main contribution of our protocol is the decoupling between the authentication and the key request, which is achieved through the use of zk-SNARKs: while requesting a key, Users must prove that they have previously performed authentication.

During the Authentication, Users must present valid credentials that certify their identity, such as a digital certificate. If those credentials are correct, they are allowed to publish a *Commitment*, which the Server includes in a structure of valid Commitments (which, for efficiency reasons, is a Merkle Tree). Thus, a Commitment identifies a specific User, because is published alongside authentication information. It also represents the User's right to obtain a QRNG key in later steps.

Users must generate a *Nullifier*, which is a value that is cryptographically related to their Commitment: they are generated from the same inputs, but only the User that has created both knows that they are related. They also generate a zk-SNARK proof that certifies the relationship between the Commitment and Nullifier, but without revealing the former. The proof also certifies that the Commitment is inside the structure of valid Commitments.

To perform the Key Request, Users send the aforementioned Nullifier and zk-SNARK proof to the Server, which validates those values. If the proof is valid, the Server (1) knows that there exists a Commitment related to this specific Nullifier inside the structure of valid Commitments, but (2) does not know which of all valid Commitments it is. We remark that (1) proves that the User has performed the Authentication exchange, while (2) ensures that neither the Server or any other entity can know the identity of the User. Finally, the Server stores the Nullifier to prevent replay attacks: any other Key Request that contains the same Nullifier will be rejected.

4.2. Architecture

In this protocol, there are three main entities:

- *User (U)*: executes the protocol and starts the communication for both Steps. In the Authentication Step, Users must provide their certificate and generate a Commitment. Then, they generate a Nullifier and a zk-SNARK linking it to the Commitment.
- *Authentication Server* (AS): receives authentication requests from Users. When a User's certificate is successfully validated, the AS inserts the provided Commitment into the Commitment Merkle Tree and updates the required nodes and root. The AS must also provide the Merkle Tree to anyone that requests it.
- *Proof Validation Server* (PVS): receives zk-SNARKs from Users and validates them. The PVS stores the list of all published Nullifiers, so it rejects any proof that is accompanied by a Nullifier that has already been published. The PVS requires information about the Merkle Tree to validate zk-SNARKs, so it must communicate with the AS to obtain this data structure. It is the only entity in the protocol that can access the QRNG, which it uses to generate key material to Users that present valid zk-SNARKs.

The AS and PVS are logical entities, so they can be implemented either as one or two different programmes. All information hosted by the AS and the PVS (namely, the Commitment Merkle Tree and the Nullifier List) is exposed so that Users can freely access it without compromising the security of the protocol.

Fig. 1 shows a diagram of the interaction between the User, Authentication Server and Proof Validation Server, as sketched in 4.1.

4.3. Protocol definition

The protocol requires the selection of a hash function H, a zk-SNARK scheme *ZK = (ZK.Setup, ZK.Prove, ZK.Verify)*, a Merkle Tree scheme *M = (M.EmptyTree, M.AddLeaf, M.GetRoot, M.GetIndexOf, M.ValidationList,M.IsLeafOfTree)* and a KEM *K = (K.KeyGen, K.Encap, K.Decap)*. Table 2 introduces the notations of the elements that take part in the protocol.

Initial setup. Before the execution of the protocol, some parameters need to be initialised. The operation executed in this initial step is:

- $\mathsf{ServerSetup}(R, n) \rightarrow ((pk_{crs}, vk_{crs}), T, T_{old}, L_N)$: The keys and data structures that will be used during the execution of the protocol are initialised. The zk-SNARK scheme is initialised from its relation R, and an empty Merkle Tree of depth n is created.

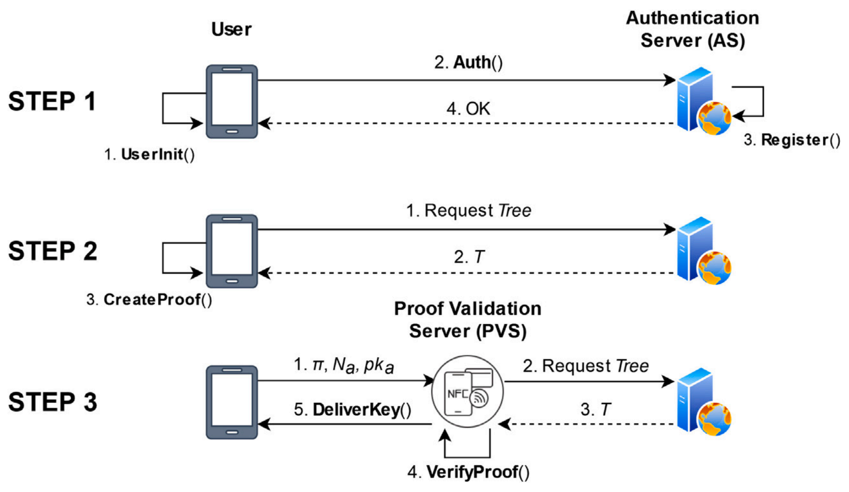

Fig. 1. Diagram of the proposed protocol.

Table 2
List of notations of the proposed scheme.

Name	Description
crs	Common reference string of zk-SNARK
pk_{crs}	Proving key of zk-SNARK
vk_{crs}	Verification key of zk-SNARK
T	Commitment Merkle Tree
T_{old}	List of previous roots of T
L_N	List of already published Nullifiers
$cert$	User certificate
ck	Private key of $cert$

Algorithm 1 Initial setup operations

```
function ServerSetup($R, n$)
    $crs \leftarrow$ ZK.Setup($R$)
    $T \leftarrow$ M.CreateEmptyTree($n$)
    $T_{old} \leftarrow \emptyset$
    $L_N \leftarrow \emptyset$
    return $crs, T, T_{old}, L_N$
end function
```

The relation that the zk-SNARK scheme is initialised with is the following:

$$R = \left\{ \begin{array}{l} (x, w) = \\ ((N, root), (\rho, pk, sk, C, i_C, Val)) \end{array} : \begin{array}{l} C = H(sk \parallel \rho), \\ N = H(pk \parallel \rho), \\ \text{M.IsLeafOfTree}(C, i_C, Val, root) \end{array} \right\} \quad (1)$$

Intuitively, it proves that the Commitment and Nullifier are generated from a key pair (pk, sk) and a secret value ρ, which is only known by the party that created both hashes. It also proves that the Commitment is inside a Merkle Tree with root $root$. This same relation is shown in Fig. 2.

Step 1: Authentication. To perform the first Step of the protocol, an User U must first create a *note*, which contains a secret value and a key pair. Then, she must authenticate herself to the AS. If she is successful, the AS will add C_U to T. The Authentication Step involves the execution of the following operations (as shown in Algorithm 2):

- UserInit(λ) $\rightarrow$ $note$: U creates a $note_U$ containing a secret value ρ_U and an asymmetric encryption key pair (pk_U, sk_U). She then stores it in a secure manner. The size of these values depends on the security parameter λ.

Algorithm 2 Authentication Step operations

```
function UserInit($\lambda$)
    $\rho \overset{R}{\leftarrow} \{0,1\}^\lambda$
    $(pk, sk) \leftarrow$ K.KeyGen($\lambda$)
    $note \leftarrow (\rho, pk, sk)$
end function
function Auth(ck, note)
    $(\rho, pk, sk) \leftarrow note$
    $C \leftarrow H(sk \parallel \rho)$
    $\sigma \leftarrow$ Sign($C, ck$)
    return $note, C, \sigma$
end function
function Register($cert, C, \sigma, T, T_{old}$)
    $b \leftarrow$ VerifySign($cert, C, \sigma$)
    if $b = true$ then
        $root \leftarrow$ M.GetRoot(T)
        $T_{old} \leftarrow T_{old} + root$
        M.AddLeaf($T, C$)
    end if
    return $b$, $T$, $T_{old}$
end function
```

- Auth($cert, ck, note$) $\rightarrow$ (C, σ_C): U uses $note_U$ to generate a Commitment $C_U = H(pk_U \parallel \rho)$. where "$\parallel$" is the concatenation operator. Then, she signs C_U with her certificate's private key ck_U.
- Register($cert, C, \sigma_C, T, T_{old}$) $\rightarrow$ $(true/false, T, T_{old})$: The AS checks the validity of the certificate $cert_U$ presented by U. Then, it verifies the signature σ_U with $cert_U$'s public key. If it is valid, the AS inserts C_U in the Merkle Tree T.

We remark that while the authentication method of the protocol we present involves the use of certificates, the Authentication Step is flexible enough to allow for other types of authentication or attestation. Different AS could be implemented that request other information from U (e.g. passwords, biometrics, …), as long as they insert C_U in T after a successful authentication.

Step 2: Proof generation. After a successful authentication, U's Commitment C_U is stored in T. In this Step, U proves that she knows some Commitment inside T by generating a zk-SNARK proof. To this end, U must first download a copy of the Merkle Tree from the AS. U can then use the Tree and $note_U$ to form the zk-SNARK Witness $w = (\rho_U, pk_U, sk_U, C_U, i_C, Val_U)$, which is needed to create the zk-SNARK proof. The following operation is executed, as shown in Algorithm 3:

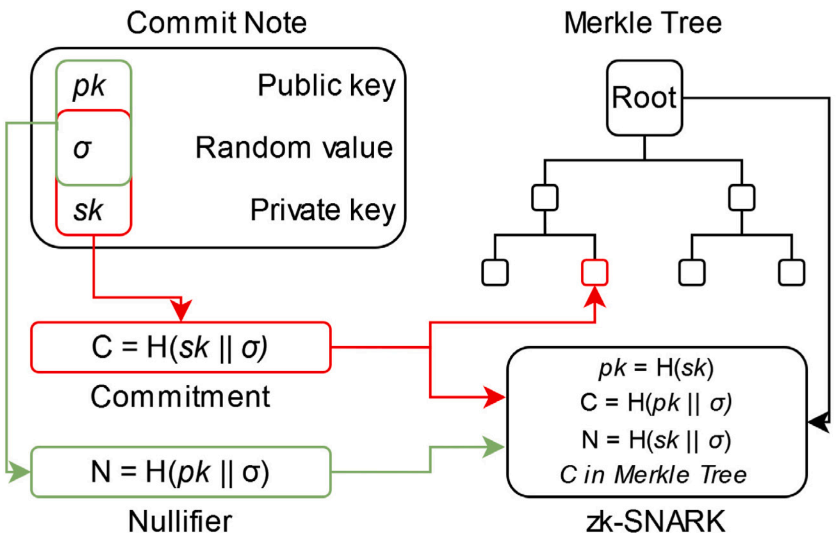

Fig. 2. Diagram of the relationship between Commitments and Nullifiers.

Algorithm 3 Proof Generation Step operations

```
function CreateProof($note, T, pk_{crs}$)
    $(\rho, pk, sk) \leftarrow note$
    $C \leftarrow H(sk \parallel \rho)$
    $N \leftarrow H(pk \parallel \rho)$
    $root \leftarrow$ M.GetRoot($T$)
    $i_C \leftarrow$ M.GetIndexOf($T, C$)
    $Val \leftarrow$ M.ValidationList($T, i_C$)
    $x \leftarrow (N, root)$
    $w \leftarrow (\rho, pk, sk, C, i_C, Val)$
    $\pi \leftarrow$ Prove($pk_{crs}, x, w$)
    return $\pi, x$
end function
```

- CreateProof$(note, T, pk_{crs}) \rightarrow (\pi, x)$: U recalculates her Commitment C_U and generates the Nullifier N_U, both of them from the values inside $note_U$. Then, she uses the zk-SNARK scheme to create a proof π_U that proves that C_U is inside T (without revealing which element it is) and that C_U and N_U were generated from the same *note*.

Step 3: Key Request. In this final Step, U provides proof that she has successfully performed Step 1 to the PVS, which is the only entity in the scheme with access to the QRNG. The message from U to the PVS includes π_U and the Statement $x = (N_U, root)$. If the information provided by U is successfully validated, the PVS will reply to U with key material generated by the QRNG.

The operations performed by U and the PVS the Key Request Step are the following (shown in detail in Algorithm 4):

- VerifyProof$(\pi, x, T, T_{old}, L_N, vk_{crs}) \rightarrow (true/false, L_N)$: The PVS verifies the zk-SNARK proof π_U generated by U in the previous phase. It also checks that the root $root_U$ that is part of the zk-SNARK statement x_U corresponds with the root of the current Merkle Tree or any of its past versions and that the Nullifier N_U has not been presented yet. If all checks are successful, N_U is included in a list of "spent Nullifiers", such that it cannot be used again.
- DeliverKey$(t, pk) \rightarrow enc$: The PVS generates t bytes of key material with its QRNG and then encrypts it with U's public key pk.

Algorithm 4 Key Request Step operations

```
function VerifyProof($\pi, x, T, L_N, T_{old}, vk_{crs}$)
    $(N, root) \leftarrow x$
    $isRootValid \leftarrow (root =$ M.GetRoot($T$) $\vee\ root \in T_{old})$
    $isNullifierValid \leftarrow (N \notin L_N)$
    $isProofValid \leftarrow$ ZK.Verify($vk_{crs}, \pi, x$)
    if $isRootValid \wedge isNullifierValid \wedge isProofValid$ then
        $L_N \leftarrow L_N + N$
        return $(true, L_N)$
    end if
    return $false, L_N)$
end function
function DeliverKey($t, pk$)
    $k \leftarrow$ GenerateQRNGMaterial($t$)
    $enc \leftarrow$ K.Encap($k, pk$)
    return $enc$
end function
```

5. Security

In this Section we will analyse the security of our privacy-preserving key transmission protocol. We informally formulate the following security requirements that the protocol must meet:

- Anonymity: it is impossible to link a User performing the Key Request Step to the credentials used in the Authentication Step.
- Unforgeability: it is impossible for an attacker to successfully perform the Key Request Step without having previously performed the Authentication Step.
- Confidentiality: the only party that can read the QRNG key created by the PVS is the User that requested it.

The properties of Anonymity and Unforgeability are common in protocols that employ zk-SNARKs (or other Zero Knowledge Proofs) as authentication mechanism [17,18,20]. We also require the property of Confidentiality because of the inclusion of a Key Encapsulation Mechanism in our protocol. Our definition of Confidentiality is similar to the KEM's property of "One Way under Chosen Plaintext Attack" [27] but applied to the environment of our protocol.

5.1. Anonymity

We will define Anonymity as a game between a Challenger C and Adversary A. In this game, C will create a pair of Commitments (C_0, C_1) and choose one of them to create a Nullifier N_b and proof π_b. A's objective is to guess which Commitment was used to generate (N_b, π_b). We will consider that the scheme provides Anonymity if A's strategy in choosing b is no better than a random guess. We remark that A is allowed to execute ServerSetup, as the Anonymity requirement also protects Users from the AS and the PVS.

Anonymity is defined as the following game:

1. A executes ServerSetup(R, n) to generate $((pk_{crs}, vk_{crs}), T_0, T_{old}, L_N)$. Then, A sends (pk_{crs}, vk_{crs}) to C.
2. C executes twice the following steps ($b \in \{0, 1\}$):
   (a) UserInit(λ) to obtain $note_b$.
   (b) From $note_b$, generate the Commitment C_b.
   (c) M.AddLeaf(T_k, C_k) to obtain T_{k+1}.
3. C randomly chooses $b \in \{0, 1\}$ and executes CreateProof$(note_b, T_2, pk_{crs})$ to obtain $(\pi_b, (N_b, root))$. C sends $((C_0, C_1), N_b, \pi_b, root)$ to A.
4. A outputs $b' \in \{0, 1\}$.

A wins the game if $b' = b$.

We define A's advantage in breaking Anonymity as

$$Adv_A^{Anon}(\lambda) = \left| \Pr[A \ wins \ the \ game] - \frac{1}{2} \right|. \tag{2}$$

Definition 5.1. The scheme provides Anonymity if for all efficient adversary A, $Adv_A^{Anon}(\lambda)$ is negligible.

Theorem 1. *If the hash function H is a one-way function, and the zk-SNARK scheme satisfies the Zero-Knowledge property, the proposed system provides Anonymity.*

Proof. We create a series of games from Game0 to Game2.

- Game0 is a real game where the adversary, A, tries breaking Anonymity in real attack scenarios.
- Game1 is the same as Game0, but the hash function H is replaced by a random oracle. Since H is preimage-resistant, Game1 is indistinguishable from Game0. This means that A cannot obtain $note_b$ from N_b or (C_0, C_1).
- Game2 is the same as Game1, but π_b is changed to a simulated π'_b instead of being generated by a real w_b. Due to the Zero-Knowledge property of the zk-SNARK, π_b and π'_b are indistinguishable and the probability of A winning Game 2 is the same as winning Game 1.

In Game2, none of the information available to A reveals anything about $note_b$: the hash functions are not reversible and the zk-SNARK does not leak any information about its witness. Thus, A cannot know if N_b was generated from the same Note as C_0 or C_1 and A's probability of correctly guessing b is no better than a random guess. Because of this, the proposed system meets the Anonymity requirement. □

5.2. Unforgeability

We will define Unforgeability as a game between Challenger C and Adversary A. In this game, A will try to generate a proof π_A without knowing the *note* from which any of the Commitments in the Merkle Tree were generated. We will consider that the system provides Unforgeability if the probability of π_A being a valid proof is negligible.

Unforgeability is defined as the following game:

1. C runs ServerSetup(R, n) to generate $((pk_{crs}, vk_{crs}), T_0, T_{old}, L_N)$. C then executes n times:
   (a) UserInit(λ) to obtain $note_k$.
   (b) From $note_k$, generate the Commitment C_k.
   (c) $T_{old} \leftarrow T_{old} +$ M.GetRoot(T_k).
   (d) M.AddLeaf(T_k, C_k) to obtain T_{k+1}.
2. C sends $(T_n, T_{old}, pk_{crs}, vk_{crs})$ to A.
3. A can execute any number of *proof generation queries*, which are defined as:
   (a) A chooses any $l \leq n$ and sends it to C.
   (b) C uses $note_l$ to execute CreateProof$(note_l, T_n, pk_{crs})$ to generate $(\pi_l, (N_l, root))$. C sends $(\pi_l, N_l, root)$ to A.
4. A generates $(\pi_A, N_A, root_A)$.

A wins if VerifyProof$(\pi_A, (N_A, root_A), T_n, T_{old}, N_0, \ldots, N_l, vk_{crs})$ outputs *true*.

We define A's advantage in breaking Unforgeability as

$$Adv_A^{Unf}(\lambda) = \Pr[A \ wins \ the \ game]. \tag{3}$$

Definition 5.2. The scheme provides Unforgeability if for all efficient adversary A, $Adv_A^{Unf}(\lambda)$ is negligible.

Theorem 2. *If the hash function H is a one-way function, and the zk-SNARK scheme satisfies the Knowledge Soundness and Weak Simulation-Extractability properties, the proposed system provides Unforgeability.*

Proof. We create a series of games from Game0 to Game5.

- Game0 is a real game where the adversary, A, tries breaking Unforgeability in real attack scenarios.
- Game1 is the same as Game0, but the hash function H is replaced by a random oracle. Since H is preimage-resistant, Game1 is indistinguishable from Game0. This means that A cannot obtain any value inside $note_k$ from any of the C_k in T_n.
- Game2 is the same as Game1, but A forges a fake Merkle Tree T_f which includes a fake Commitment C_f. Since H is a random oracle, the root of T_f will be different than any T_k's root because their leaves are different. Thus, Condition (2) is not met and A does not gain any advantage: Game2 is indistinguishable from Game1.
- Game3 is the same as Game2, but A creates a Fake Commitment C_f and forges a fake validation list Val_f leading to the root of any T_k. From this, A will generate a proof π_f. This requires finding a collision, in which two different values have the same hash (the root of T_k). Since H is a random oracle, it will be impossible to find a collision that would falsely verify that C_f is in T_k. Thus, the relation described in Eq. (1) will not hold and the Knowledge Soundness property of the zk-SNARK scheme will ensure that Condition (3) is not met. Thus, Game3 is indistinguishable from Game2.
- Game4 is the same as Game3, but A is able to generate a zk-SNARK π_A without knowing any of the values in the witness $w_A = (\rho_A, pk_A, sk_A, C_A, i_C, Val_C)$. Due to the Knowledge Soundness property of the zk-SNARK, Condition (3) will not be met with overwhelming probability. Therefore, Game4 does not increase A's probability of winning over Game3.
- Game5 is the same as Game4, but A is able to forge a fake proof π_f by modifying a zk-SNARK π_l received during Step 4 of the attack game (i.e. if the zk-SNARK scheme is vulnerable to malleability attacks [24]). There are two possibilities in this case:

– A presents (π_f, N_l), that is, A does not modify the Nullifier. In this case, A fails because of Condition (1).
– A modifies N_l into a different N_f. Due to the Weak Simulation-Extractability property of the zk-SNARK scheme, the output of Verify will be $Reject$, since it is assumed to be impossible to generate a valid proof for a modified statement. Thus, A fails because of Condition (3).

In both cases, the probability of A winning Game5 is the same as winning Game4.

In Game5, the probability achieving any of the conditions required to break the Unforgeability requirement is negligible. Since Game0 and Game5 are indistinguishable from each other, the proposed protocol meets the Unforgeability requirement. □

5.3. Confidentiality

We will define Confidentiality as a game between Challenger C and Adversary A. In this game, C will generate a Commitment and Nullifier pair from a $note$, and an encrypted QRNG key using the public key in said $note$. We will consider that the system provides Confidentiality if A's strategy in finding the QRNG key is no better than a random guess.

Confidentiality is defined as the following game:

1. A executes ServerSetup(R, n) to generate $((pk_{crs}, vk_{crs}), T_0, T_{old}, L_N)$. Then, A sends (pk_{crs}, vk_{crs}) to C.
2. C executes the following steps:
   (a) UserInit(λ) to obtain $note = (\rho, pk, sk)$.
   (b) From $note$, generate the Commitment C.
   (c) M.AddLeaf(T_0, C) to obtain T_1.
   (d) CreateProof$(note, T_1, pk_{crs})$ to obtain $(\pi, (N, root))$
   (e) DeliverKey(t, pk) to generate enc.
3. C sends (C, N, π, pk, t, enc) to A.
4. A outputs y.

A wins if $y = K.Decap(enc, sk)$.

We define A's advantage in breaking Confidentiality as

$$Adv_A^{Conf}(\lambda, t) = \left| \Pr[A \ wins \ the \ game - \frac{1}{2^t}] \right|. \quad (4)$$

Definition 5.3. The scheme provides Confidentiality if for all efficient adversary A, $Adv_A^{Conf}(\lambda, t)$ is negligible.

Theorem 3. *If the hash function H is a one-way function, the zk-SNARK scheme satisfies the Zero-Knowledge property and the Key Encapsulation mechanism is a one-way trapdoor function scheme, the proposed system provides Confidentiality.*

Proof. We create a series of games from Game0 to Game3.

• Game0 is a real game where the adversary, A, tries breaking Confidentiality in real attack scenarios.
• Game1 is the same as Game0, but the hash function H is replaced by a random oracle. Since H is preimage-resistant, Game1 is indistinguishable from Game0. This means that A cannot obtain sk from N or C.
• Game2 is the same as Game1, but π is changed to a simulated π' instead of being generated by a real w. Due to the Zero-Knowledge property of the zk-SNARK, π and π' are indistinguishable and the probability of A winning Game2 is the same as winning Game1. Since π' was not generated by a real w, it is impossible to obtain sk from π'. Thus, none of the values received by A reveal anything about sk.
• Game3 is the same as Game2, but the Key Encapsulation Method' Encap function is replaced by a random oracle. Since in Game2 A does not know the trapdoor sk, Game3 is indistinguishable from Game2.

In Game3, the value enc received by A is random and not related to the QRNG-generated key y. Since Game3 is indistinguishable from Game0, the proposed system meets the Confidentiality requirement. □

6. Implementation

We provide an implementation of our protocol to demonstrate its applicability. We represent a real use case in which the QRNG keys are transmitted through NFC to the User's mobile device, such that these keys could be used to securely encrypt the User's information inside their phone. Our implementation is composed of the following elements:

• Three Java programmes representing the User, Authentication Server and Proof Validation Server. Both servers are located in the same machine, and have local access to their shared files. The AS is a REST API implemented with Spring Boot. Since the servers do not need to keep any information private, the AS allows Users to request any of its data structures: Merkle Tree, Nullifier List and the list of old Merkle Roots.
• A library tasked with the implementation of the zk-SNARK scheme, including circuit definition and proof generation and validation.
• A QRNG device for the generation of the key material.
• An NFC tunnel for the communication between the User and the PVS. To this end, we have implemented an Android App that employs the User code and acts as an NFC Smart Card. The PVS has access to an *ACR 122U USB* NFC Reader to receive requests from Users.

6.1. zk-SNARKs

All operations related to zk-SNARK generation and validation in this work have been implemented with two different libraries: libsnark [30] and the ZoKrates toolbox [31], which will be referred as *zk-SNARK backends*. Both of these libraries require defining the statements that the zk-SNARK should prove, which in this protocol correspond to the statements that were specified in Section 4.

Both zk-SNARK backends require a similar process for the definition of the zk-SNARK scheme. First, the relation R (in this case, the relation shown in Eq. (1)) must be codified in the zk-SNARK backend's specific language. Then, this code is compiled and an arithmetic circuit representing the calculation is generated. For both zk-SNARK backends, Users can interact with the compiled programme through the command line. Once compiled, the Common Reference String (CRS) can be generated from the arithmetic circuit. The CRS is composed of the Proving Key and the Verification Key. To execute Verify, Users must provide both the Statement $x = (root, N)$ and the Witness $w = (\rho, pk, sk, C, i_C, Val)$. To Verify a proof π, only the Statement is required.

We have chosen Groth16 [23] as the zk-SNARK scheme, since it achieves the fastest proof generation and validation times [24]. Moreover, it satisfies all properties mentioned in Section 3.1 [32]. We remark that the proposed protocol only requires the notion of Weak Simulation-Extractability to provide Unforgeability.

Since both zk-SNARK backends require knowing the number of parameters in compilation time, the Commitment Merkle Tree must be of fixed depth, limiting the maximum number of possible Commitments. The impact of this restriction will be studied in Section 7.1.

6.2. QRNG

For this work, two different QRNGs have been employed:

• *IDQuantique's "Quantis QNRG USB"*: This device is accessible via an USB interface, and its manufacturers provide an API for multiple programming languages. Its key generation rate is about 4 Mb/s.

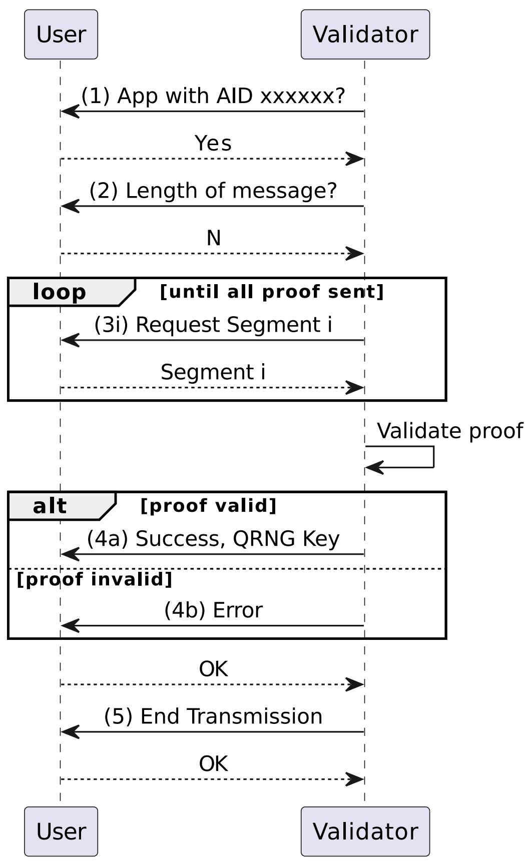

Fig. 3. Sequence diagram of the NFC tunnel.

- *Centro de Supercomputación de Galicia's (CESGA) QRNG module*: This supercomputer has a native QRNG infrastructure that can generate keys at 400 Mb/s. A remote SSH connection is required to gain access to this high-rate source, which introduces an overhead for establishing the connection and transmitting the key material.

6.3. NFC tunnel

Near Field Communication (NFC) [33–36] is a technology that allows wireless communication between two physically adjacent devices: an active Reader (which must always initiate the communication) and a passive Smart Card (which can only respond to the Reader's messages). NFC messages are encapsulated in Application Protocol Data Units (APDUs).

NFC is considered more secure than other wireless communication technologies such as Bluetooth because of its very small area of transmission (which prevents eavesdroppers from capturing APDUs) and its fast setup phase. Most mobile devices currently support NFC, with the capability of simulating a Smart Card and dynamically creating APDUs to respond to NFC readers. This makes NFC a very comfortable way to distribute key material to users, as it can be stored in their mobile device.

In our implementation, all communication between Users and the PVS in Step 3 is performed through NFC. The PVS' NFC reader interacts with U's mobile device, which contains an App that can execute our protocol. Since the reader must always initiate the communication, an specific scheme is required for transmitting the zk-SNARK proof from the User to the PVS, which is shown in 3.

The PVS must first make sure that U's mobile device contains the corresponding App. If correct, the PVS asks for the length of the message to be sent, and then performs as many requests as needed to receive the complete proof. The proof will be divided into segments to account for the small size of APDUs, which is usually limited to 255 bytes. The PVS then checks if the proof is valid and sends the QRNG if correct or an error otherwise. Finally, the PVS sends an "End Transmission" message to signal that both agents can free the employed resources.

Table 3
Parameters of the simulation environment.

Parameter	Value
Operating system	Windows 10
CPU	Intel Core i7-10700
Clock speed	2.90 GHz
Memory	32 GB
zk-SNARK tool	ZoKrates 0.8.3; libsnark
zk-SNARK scheme	Groth16
Programming language	Java 17 (User, AS, PVS); ZoKrates, C++ (zk-SNARKs)
Hash function	SHA-256
KEM	RSA-OAEP

7. Experimental results

In this Section we analyse the performance of the implementation introduced in the previous Section. All of the following tests have been executed in a simulation environment, whose properties are shown in Table 3.

7.1. Proof generation

We will measure how proof generation time is influenced by the depth of the Merkle Tree. This value determines the maximum amount of Commitments it can store. However, increasing this value also increases the amount of hashes during the execution of M.IsLeafOfTree. As mentioned in Section 5, the efficiency of proof generation is heavily dependent on the number of constraints of the arithmetic circuit. The function M.IsLeafOfTree is part of the relation R shown in Eq. (1), so increasing the depth of the Merkle Tree will negatively impact the proof generation time.

Fig. 4 shows the influence of the Merkle Tree depth proof generation time. Validation time has been omitted as it remained constant around 30 ms. The influence of the Merkle Tree depth has been tested for both zk-SNARK backends, of which libsnark achieves lower proof generation times. ZoKrates starts performing noticeably worse when Merkle Tree depth becomes 19 or higher. While Tree size (i.e., the amount of Commitments that can be stored) increases exponentially with depth, proof generation time increases only linearly.

7.2. Key transmission

The Key Request Step can be divided into four different actions: transmission of the zk-SNARK proof from U to the PVS, proof validation, QRNG key generation and the transmission of the QRNG key from the PVS to U. Since proof size and validation time are constant according to the definition of zk-SNARKs, only QRNG key generation and transmission are variable.

We measured the total duration of the Key Request Step for different requested key sizes. Results are shown in Fig. 5: when key sizes are small, most of the time is spent transmitting the zk-SNARK and performing validation. As keys grow in size, key transmission becomes an important factor, as NFC has a relatively small transmission rate. When requesting keys of 4096 bytes in size, around 60 per cent of time is spent on transmitting the QRNG key over NFC.

Key generation time is also influenced by requested key size. Quantis' QRNG and CESGA's QRNG are compared in Fig. 6. CESGA's time has been divided into two categories: "Generation" and "SSH". The latter

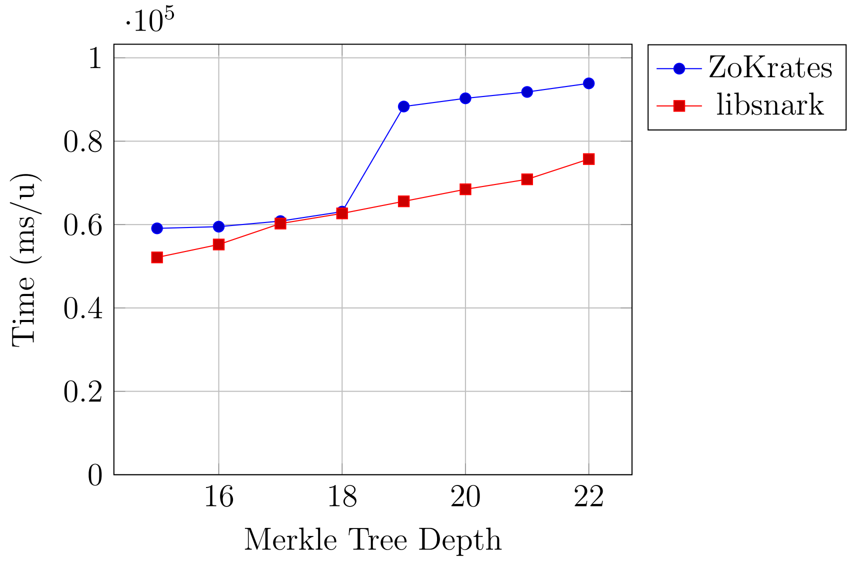

Fig. 4. Proof generation times for different Merkle Tree depths.

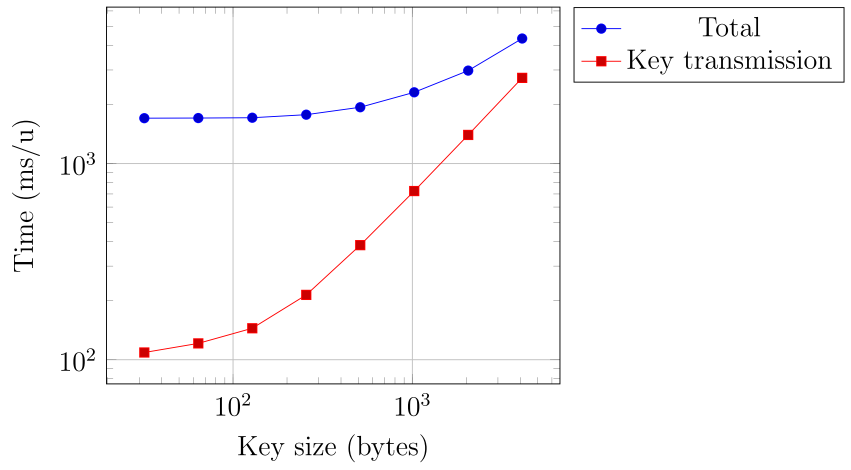

Fig. 5. Key Request Step time, with QRNG key transmission being also shown for comparison.

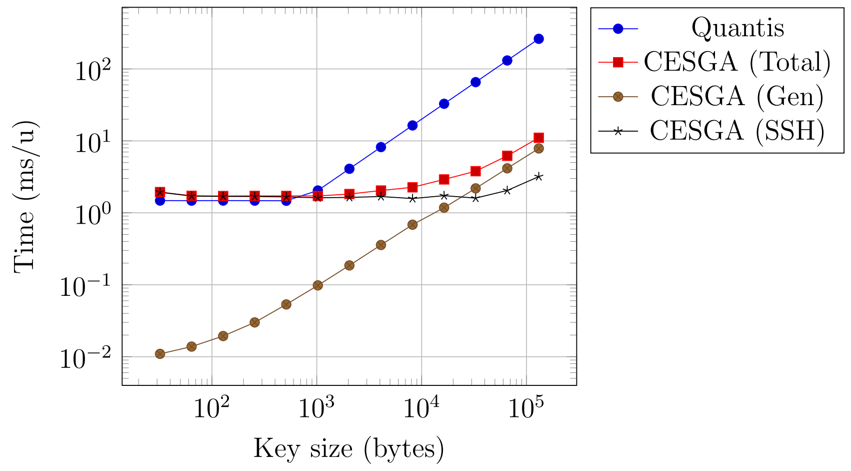

Fig. 6. Key generation time for both QRNGs.

corresponds to the time spent in establishing a remote connection with the QRNG and transmitting the generated key material back to the PVS.

For small key sizes, Quantis performs better than CESGA, because most of CESGA's time is spent in establishing a remote SSH connection. CESGA's higher key generation rate becomes noticeable when requested key size increases, and SSH tunnel establishing time becomes less relevant. SSH time also includes transmitting the key from CESGA's QRNG to the PVS, so it starts to increase when requesting large amounts of key material.

We remark that zk-SNARK proof generation, which represents the most expensive computational workload of our protocol, is performed offline by the Users and thus can be executed by any number of them at the same time. The AS and PVS can handle a large amount of concurrent Users: as shown in Figs. 5 and 6, the time costs of the Authentication and Key Request Steps are negligible compared to that of the Proof Generation Step.

8. Conclusion and future work

This work has presented a privacy-preserving key transmission protocol for QRNG-generated cryptographic material, in which zk-SNARKs are employed to ensure that the server is oblivious of the authenticated Users requesting each key. Consequently, users do not have to trust the servers to obtain secure keys. The protocol could be employed to provide high-entropy key material to end users, which they could plug into their communications protocol stack or their applications, but we emphasise that the protocol is totally agnostic of the source used to generate the keys. For instance, we can envision our solution as a component in a QKD metropolitan network. In this architecture, a predefined number of nodes are connected through QKD channels such that any pair of points can securely establish a shared secret between them. Since access to these QKD nodes could be very restricted, and effectively only available for high-security entities or application domains, the QKD keys would have to be distributed outside the quantum-enabled domain in an anonymous and protected form to extend the reach of QKD. To that end, an algorithmic solution like the one proposed in this paper can be useful.

We provide formal security proofs to the properties of Anonymity, Unforgeability and Confidentiality. Thus, we have proven that performing the Key Request Step does not reveal any information about which credentials were used in the Authentication Step, it is not possible to create a valid proof without first performing the Authentication Step, and only the User that performed the Key Request Step can access the QRNG key. Our implementation of the proposed protocol demonstrates its applicability to a real use case, in which the QRNG-generated keys are directly deployed to the Users' mobile phone through NFC using a custom method for transmitting arbitrary data encapsulated in APDUs We have implemented the zk-SNARK generation and validation operations with two different libraries and measure their performance under different circumstances. We plan to further develop our proof-of-concept Android Application to provide real functionality for the QRNG-generated key material.

CRediT authorship contribution statement

David Soler: Conceptualization, Formal analysis, Investigation, Software, Writing – original draft. **Carlos Dafonte:** Conceptualization, Supervision, Writing – review & editing. **Manuel Fernández-Veiga:** Conceptualization, Supervision, Writing – review & editing. **Ana Fernández Vilas:** Conceptualization, Supervision, Writing – review & editing. **Francisco J. Nóvoa:** Conceptualization, Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgements

The work is funded by the Plan Complementario de Comunicaciones Cuánticas, Spanish Ministry of Science and Innovation(MICINN), Plan de Recuperación NextGeneration, European Union (PRTR-C17.I1, CITIC Ref. [305].2022), and Regional Government of Galicia (Agencia Gallega de Innovación, GAIN, CITIC Ref. [306].2022) D.S. acknowledges support from Xunta de Galicia and the European Union (European Social Fund - ESF) scholarship [ED481A-2023-219].

This work is part of the project TED2021-130369B-C31 and TED2021-130492B-C21 funded by MCIN/AEI/10.13039/501100011033 and by "ERDF A way of making Europe".

This work has been developed thanks to the access granted by the Centro de Supercomputación de Galicia to the infrastructure based in quantum technologies. This infrastructure was financed by the European Union, through the FONDO EUROPEO DE DESENVOLVEMENTO REXIONAL (FEDER), as part of the Union's response to the COVID-19 pandemic.

References

[1] K. Bhattacharjee, S. Das, A search for good pseudo-random number generators: Survey and empirical studies, Comput. Sci. Rev. 45 (C) (2022) http://dx.doi.org/10.1016/j.cosrev.2022.100471.

[2] A. Saini, A. Tsokanos, R. Kirner, Quantum randomness in cryptography—a survey of cryptosystems, RNG-based ciphers, and QRNGs, Information 13 (8) (2022) 358, http://dx.doi.org/10.3390/info13080358.

[3] N. Heninger, Z. Durumeric, E. Wustrow, J. Halderman, Mining your Ps and Qs: detection of widespread weak keys in network devices, in: 21st USENIX Security Symposium (USENIX Security 12), 2012, pp. 35–35.

[4] D. Antonioli, N.O. Tippenhauer, K.B. Rasmussen, The KNOB is broken: Exploiting low entropy in the encryption key negotiation of bluetooth BR/EDR, in: 28th USENIX Security Symp. (USENIX Security 19), Santa Clara, CA, 2019, pp. 1047–1061.

[5] D. Kaplan, S. Kedmi, R. Hay, A. Dayan, Attacking the Linux PRNG on android: Weaknesses in seeding of entropic pools and low boot-time entropy, in: Proc. of the 8th USENIX Conference on Offensive Technologies, WOOT'14, USA, 2014, p. 14.

[6] D. Hurley-Smith, J. Hernandez-Castro, Quantum leap and crash: Searching and finding bias in quantum random number generators, ACM Trans. Priv. Secur. 23 (3) (2020) 1–25, http://dx.doi.org/10.1145/3398726.

[7] M. Stipčević, Quantum random number generators and their use in cryptography, in: 2011 Proceedings of the 34th International Convention MIPRO, 2011, pp. 1474–1479.

[8] M.M. Jacak, P. Jóźwiak, J. Niemczuk, J.E. Jacak, Quantum generators of random numbers, Sci. Rep. 11 (1) (2021) 16108, http://dx.doi.org/10.1038/s41598-021-95388-7.

[9] A. Vassilev, R. Staples, Entropy as a service: Unlocking cryptography's full potential, Computer 49 (9) (2016) 98–102, http://dx.doi.org/10.1109/MC.2016.275.

[10] A. Kavousi, Z. Wang, P. Jovanovic, Sok: Public Randomness, Paper 2023/1121, Cryptology ePrint Archive, 2023, URL https://eprint.iacr.org/2023/1121.

[11] M. Raikwar, D. Gligoroski, Sok: Decentralized randomness beacon protocols, in: Australasian Conference on Information Security and Privacy, Springer, 2022, pp. 420–446.

[12] M. Haahr, Random. Org: True Random Number Service, School of Computer Science and Statistics, Trinity College, Dublin, Ireland, 2010, Website (http://www.random.org). Accessed 10 (2010).

[13] M. Agal, K.P. Kishan, R. Shashidhar, S.S. Vantmuri, P. Honnavalli, Non-interactive zero-knowledge proof based authentication, in: 2021 IEEE Mysore Sub Section International Conference (MysuruCon), 2021, pp. 837–843, http://dx.doi.org/10.1109/MysuruCon52639.2021.9641514.

[14] F. Martín-Fernández, P. Caballero-Gil, C. Caballero-Gil, Authentication based on non-interactive zero-knowledge proofs for the internet of things, Sensors 16 (1) (2016) http://dx.doi.org/10.3390/s16010075.

[15] M. Walshe, G. Epiphaniou, H. Al-Khateeb, M. Hammoudeh, V. Katos, A. Dehghantanha, Non-interactive zero knowledge proofs for the authentication of IoT devices in reduced connectivity environments, Ad Hoc Netw. 95 (2019) 101988, http://dx.doi.org/10.1016/j.adhoc.2019.101988.

[16] S. Liu, L. Chen, H. Yu, S. Gao, H. Fang, BP-AKAA: Blockchain-enforced privacy-preserving authentication and key agreement and access control for iiot, J. Inf. Secur. Appl. 73 (2023) 103443, http://dx.doi.org/10.1016/j.jisa.2023.103443.

[17] H. An, J. Chen, Elearnchain: A privacy-preserving consortium blockchain system for e-learning educational records, J. Inf. Secur. Appl. 63 (2021) 103013, http://dx.doi.org/10.1016/j.jisa.2021.103013.

[18] D.A. Luong, J.H. Park, Privacy-preserving blockchain-based healthcare system for iot devices using zk-snark, IEEE Access 10 (2022) 55739–55752, http://dx.doi.org/10.1109/ACCESS.2022.3177211.

[19] J. Lee, J.Y. Hwang, J. Choi, H. Oh, J. Kim, Sims : Self sovereign identity management system with preserving privacy in blockchain, iacr cryptol, ePrint Arch. 2019 (2019) 1241.

[20] P. Li, J. Lai, Y. Wu, Event-oriented linkable and traceable anonymous authentication and its application to voting, J. Inf. Secur. Appl. 60 (2021) 102865, http://dx.doi.org/10.1016/j.jisa.2021.102865.

[21] D. Hopwood, S. Bowe, T. Hornby, N. Wilcox, Zcash Protocol Specification, vol. 4, GitHub, San Francisco, CA, USA, 2016, p. 220.

[22] H. Ko, I. Lee, S. Lee, J. Kim, H. Oh, Efficient Verifiable Image Redacting Based on Zk-Snarks, Paper 2020/1579, Cryptology ePrint Archive, 2020, http://dx.doi.org/10.1145/3433210.3453110.

[23] J. Groth, On the size of pairing-based non-interactive arguments, in: M. Fischlin, J.-S. Coron (Eds.), Advances in Cryptology – EUROCRYPT 2016, Springer Berlin Heidelberg, Berlin, Heidelberg, 2016, pp. 305–326.

[24] K. Baghery, Z. Pindado, C. Ràfols, Simulation extractable versions of groth's zk-snark revisited, in: S. Krenn, H. Shulman, S. Vaudenay (Eds.), Cryptology and Network Security, 2020, pp. 453–461.

[25] R.C. Merkle, A digital signature based on a conventional encryption function, in: C. Pomerance (Ed.), Advances in Cryptology — CRYPTO '87, Springer Berlin Heidelberg, Berlin, Heidelberg, 1988, pp. 369–378.

[26] C. Coronado, On the security and the efficiency of the Merkle signature scheme, IACR cryptol, ePrint Arch. 2005 (2005) 192.

[27] M. Campagna, A. Petcher, Security of Hybrid Key Encapsulation, Paper 2020/1364, Cryptology ePrint Archive, 2020.

[28] R. Barnes, K. Bhargavan, B. Lipp, C. Wood, Hybrid public key encryption, RFC 9180, RFC editor, 2022.

[29] J. Alwen, B. Blanchet, E. Hauck, E. Kiltz, B. Lipp, D. Riepel, Analysing the Hpke Standard, Paper 2020/1499, Cryptology ePrint Archive, 2020, http://dx.doi.org/10.1007/978-3-030-77870-5_4.

[30] S. Lab, Libsnark: a C++ library for zk-SNARK proofs, 2020, https://github.com/scipr-lab/libsnark.

[31] J. Eberhardt, S. Tai, Zokrates - scalable privacy-preserving off-chain computations, in: 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), 2018, pp. 1084–1091, http://dx.doi.org/10.1109/Cybermatics_2018.2018.00199.

[32] K. Baghery, M. Kohlweiss, J. Siim, M. Volkhov, Another Look At Extraction and Randomization of Groth's Zk-Snark, Paper 2020/811, Cryptology ePrint Archive, 2020.

[33] V. Coskun, B. Ozdenizci Kose, K. Ok, A survey on near field communication (NFC) technology, Wirel. Pers. Commun. 71 (2013) http://dx.doi.org/10.1007/s11277-012-0935-5.

[34] E. Haselsteiner, K. Breitfuss, Security in Near Field Communication (NFC), Workshop on RFID Security, 2006.

[35] G. Madlmayr, J. Langer, C. Kantner, J. Scharinger, NFC devices: Security and privacy, in: 2008 Third International Conference on Availability, Reliability and Security, 2008, pp. 642–647, http://dx.doi.org/10.1109/ARES.2008.105.

[36] M.M. Mahinderjit Singh, K. Adzman, R. Hassan, Near field communication (NFC) technology security vulnerabilities and countermeasures, Int. J. Eng. Technol. 7 (2018) 298–305, http://dx.doi.org/10.14419/ijet.v7i4.31.23384.

David Soler received his B.S degree in Computer Engineering in University of Santiago de Compostela, Galicia, Spain and his M.S degree in Cybersecurity in University of A Coruna in 2021 and 2023, respectively. He is currently a Ph.D student in University of A Coruna. His main interests include cybersecurity, applied cryptography, and network communications.