Public-Key Generation with Verifiable Randomness

Olivier Blazy[1], Patrick Towa[2,3], Damien Vergnaud[4,5]

[1] Universite de Limoges
[2] IBM Research – Zurich
[3] DIENS, École Normale Supérieure, CNRS, PSL University, Paris, France
[4] Sorbonne Université, CNRS, LIP6, F-75005 Paris, France
[5] Institut Universitaire de France

Abstract. We revisit the problem of proving that a user algorithm selected and correctly used a truly random seed in the generation of her cryptographic key. A first approach was proposed in 2002 by Juels and Guajardo for the validation of RSA secret keys. We present a new security model and general tools to efficiently prove that a private key was generated at random according to a prescribed process, without revealing any further information about the private key.
We give a generic protocol for all key-generation algorithms based on probabilistic circuits and prove its security. We also propose a new protocol for factoring-based cryptography that we prove secure in the aforementioned model. This latter relies on a new efficient zero-knowledge argument for the double discrete logarithm problem that achieves an exponential improvement in communication complexity compared to the state of the art, and is of independent interest.

1 Introduction

Cryptographic protocols are commonly designed under the assumption that the protocol parties have access to perfect (i.e., uniform) randomness. However, random sources used in practical implementations rarely meet this assumption and provide only a stream of bits with a certain "level of randomness". The quality of the random numbers directly determines the security strength of the systems that use them. Following preliminary work by Juels and Guajardo [22] and Corrigan-Gibbs, Mu, Boneh and Ford [15], we revisit the problem of proving that a cryptographic user algorithm has selected and correctly used a truly random seed in the generation of her cryptographic public–secret key pair.

Related Work. A prominent example that the use of randomness in public-key cryptography (and especially in key-generation protocols) is error-prone is the recent randomness failure known as the *ROCA vulnerability* [25]. This weakness allows a private key to be recovered efficiently from the public key only (in factoring-based cryptography). The flawed key-generation algorithm selects specific prime numbers as part of the private key instead of generating uniformly random primes and many certified devices were shown vulnerable. This kind of weaknesses is not new as in 2012, Lenstra, Hughes, Augier, Bos, Kleinjung and Wachter [23] did a sanity check of factoring-based public keys collected on the web. They showed that a significant percentage of public keys (0.5%) share a common prime factor, and this fact was explained [21] by the generation of these low entropy keys during booting. Since cryptographic failures due to weak randomness can be dire [25,23,21], designers should build schemes that can withstand deviations of the random sources from perfect randomness.

Following seminal works by Simmons on the threat of covert channels (also called subliminal channels) in cryptography [29], the concept of *kleptography* was proposed by Young and Yung [32]. It models the fact that an adversary may subvert cryptographic algorithms by modifying their implementations in order to leak secrets using for instance covert channels present in the randomized algorithms. Several sources have recently revealed that cryptographic algorithms have effectively been subverted to undermine the security of users. This raises the concern of guaranteeing a user's security even when she may be using a compromised machine or algorithm. For factoring-based public-key cryptography, in light of the known shortcomings of implemented key generators, a line of research has focused on proving that RSA moduli satisfy certain properties [18,11,1], or on attesting that RSA prime factors were generated with a specified prime generator [5]. This line of work is only concerned with the structure of the keys, not with the fact that they are generated with enough entropy. Juels and Guajardo [22] suggested as early as in 2002 an approach for users to prove to another party (which is typically a trusted certificate authority or CA) that her public–secret key pair was generated honestly using proper randomness. In their setting, the CA provides an additional source of randomness in an interactive process, and the user algorithm proves that it has not weakened, whether intentionally or unintentionally, the key-generation procedure.The security goal of such a primitive is threefold.

1. **Maintain User Privacy:** if the user uses a randomness source with high entropy, then an adversary (possibly the CA himself) has no additional information on the secret-key compared to a key generated by the real key-generation algorithm on uniform randomness.
2. **Improve Randomness Quality:** if the user *or* the CA use a randomness source with high entropy, then, an adversary (other than the CA) has no additional information on the secret-key compared to a key generated by the real key-generation algorithm on uniform randomness.
3. **Resist Information Exfiltration:** the generated public key leaks no information whatsoever to the outer world. In particular, a faulty user algorithm cannot use it to convey any information. In this sense, the CA certifies to the end user, that she can securely use to the generated key.

A malicious user can obviously publish her secret key, but the problem we tackle is different: we want the CA to only certify keys that he knows to have been generated with high-entropy randomness and without covert channels.

Juels and Guajardo proposed a formal security model for *verifiable random key generation* with the goal to achieve these three security objectives. Their model is unfortunately not strong enough to capture real-world threats since

– it is restricted to public-key cryptosystems where a given public key corresponds to a *unique* secret key (and cannot be used for many recent schemes);
– it considers only a stand-alone or independent key-generation instances (and therefore does not prevent attacks such as the one considered in [23,21] where several public-keys are generated with correlated randomness sources);
– it only bounds the distance that a dishonest user algorithm can generate a key to that of an honest algorithm executing the key generation protocol.

As a simple example, consider the problem of generating an ElGamal public key g^x in a group $\mathbb{G} = \langle g \rangle$ of prime order p. Juels and Guajardo outlined a protocol for generating such a key with verifiable randomness. The natural idea to generate a public-key g^x in this (illusorily) simple setting is to share the secret key x as $x = x_U + x_{CA} \bmod p$ where x_U denotes the user randomness and x_{CA} denotes the CA randomness. However, this protocol fails to achieve (3) as the user algorithm can choose x_U to match a specify value after seeing x_{CA}. To overcome this issue, a simple idea would be to make the user first commit to x_U and then prove its knowledge. However, the hiding and zero-knowledge properties of commitment schemes and proof systems inherently rely on perfect randomness, which the user algorithm is assumed not to have at its disposal.

Juels and Guajardo also proposed a protocol for the generation of RSA keys where the distance in (3) increases by a factor which is polynomial in the security parameter λ (assuming some number-theoretic conjecture). Therefore, their protocol does not rule out the existence of covert channels with $O(\log \lambda)$ bit capacity. Their model was reconsidered by Corrigan-Gibbs, Mu, Boneh and Ford [15] in a weaker setting that guarantees (1) and (2) but not (3), and does not even prevent a malicious user algorithm from generating malformed keys.

Contributions. We revisit the verifiable key-generation primitive and provide the first strong security models and efficient, provably secure constructions.

Game-Based Security Model. We propose a game-based model that covers concurrent protocol executions with different instances of protocol algorithms. It is inspired by the Bellare-Pointcheval-Rogaway (BPR) model for authenticated key exchange [4]. The communication between the user and the CA is assumed to be carried over an insecure channel. Messages can be tapped and modified by an adversary, and the communication between the user and the CA is asynchronous. The adversary is split into two algorithms: (1) the *sampler* which provides the randomness sources to the user and the CA (for multiple instances of the protocol) and (2) the *distinguisher* which tries to gain information from the generated public key. The protocol is deemed secure if the distinguisher is unable to do so assuming that the entropy of either random source is high enough.

The main difficulty to define the security model for this primitive is to formalize the third security objective. A dishonest user algorithm can indeed always execute several instances of the protocol with the CA until she obtains a public-key which has some specific property which allows to exfiltrate information. This is similar to the "halting attack" subliminal channel [16] and cannot be avoided. We manage to take this *narrow-band* subliminal channel into consideration in our security model while capturing the fact that in a secure protocol, this should be the only possible covert channel for a dishonest user algorithm. In practical applications, this covert channel can be prevented easily if the CA charges an important fee for a user that performs too many key generation procedures, or if an increasing time-gating mechanism for repeating queries is introduced.

This model does not suffer from the shortcomings of the model proposed in [22] as it allows for multiple dependent runs of the protocol and captures the resistance to exfiltration of information (with only the narrow-band subliminal channel from the "halting attack"). It guarantees security with concurrent sessions (and is thus much stronger than security considered in the similar notion of cryptographic reverse firewalls [24]) but not composition.

Providing a universal-composability definition seems natural in this setting, but the main hurdle in doing so comes from the fact that the sampler cannot communicate at all with the distinguisher since it would otherwise allow for covert channels (and break property (3)) as further explained in Section 3.2. As a consequence, a universal-composability definition would need functionalities with local adversaries, which would change the target of the paper.

Generic Protocol for Probabilistic Circuits. We then present a generic approach for key generation based on (families of) probabilistic circuits and we prove its security in our stringent security model. It relies on two-source randomness extractors, pseudo-random-function families and extractable commitments with associated zero-knowledge proofs. Since two-party computation (2PC) protocols rely on perfect randomness, a generic 2PC protocol cannot be used in this setting; moreover such a protocol guarantees privacy and correctness, but it does not guarantee that a user cannot influence the result (and thus requirement (3)).

Efficient Protocol for RSA Keys. We also propose a new generic protocol for factoring-based cryptography and prove it secure in our model. It relies on classical cryptographic tools (namely commitments, pseudo-random functions (PRFs) and zero-knowledge proofs). We provide an instantiation based on the Dodis–Yampolskiy PRF [17] in the group of quadratic residue modulo a safe prime which outputs group elements. The main technical difficulty is to convert the outputs of this PRF into integers while proving that the RSA prime factors are outputs of the PRF. In the process, we propose a new efficient zero-knowledge proof system for the so-called *double discrete logarithm problem* (in groups of public order). A double discrete logarithm of an element $y \neq 1_{\mathbb{G}}$ in a cyclic group $\mathbb{G}$ of prime order p with respect to bases $g \in \mathbb{G}$ and $h \in \mathbb{Z}_p^*$ (generators of $\mathbb{G}$ and $\mathbb{Z}_p^*$ respectively) is an integer $x \in \{0, \ldots, p-1\}$ such that $y = g^{h^x}$. Stadler introduced this computational problem for verifiable secret-sharing [30] and it was used to design numerous cryptographic protocols (e.g. group signatures [12], e-cash systems [13] and credential systems [14]). All these constructions rely on a proof system proposed by Stadler which has $\Omega(\log p)$ computational and communication complexity (in terms of group elements). Our new proof system outputs proofs with only $O(\log\log p)$ group elements and permits an efficient instantiation of our generic protocol for factoring-based cryptography. As a by-product, our new protocol can be used directly in all the aforementioned applications in a public-order setting to exponentially reduce their communication complexity.

2 Preliminaries

Notation. For $n \in \mathbb{N}$, the set of n-bit strings is denoted by $\{0,1\}^n$ and the set of integers $\{1,\ldots,n\}$ is denoted $[\![n]\!]$. The set of prime numbers is denoted $\mathbb{P}$. The security parameter is denoted λ, and input lengths are always assumed to be bounded by some polynomial in λ. A Probabilistic algorithm is said to run in Polynomial-Time (it is said to be a PPT algorithm) if it runs in time that is polynomial in λ. A function μ is negligible if $\mu(\lambda) = \lambda^{-\omega(1)}$.

The random variable defined by the value returned by a PPT algorithm $\mathcal{A}$ on input x is denoted $\mathcal{A}(x)$. The value returned by $\mathcal{A}$ on input x and random string r is denoted $\mathcal{A}(x;r)$. Given a probability distribution S, a PPT algorithm that samples a random element according to S is denoted by $x \leftarrow_\$ S$. For a finite set X, $x \leftarrow_\$ X$ denotes a PPT algorithm that samples an element uniformly at random from X. Given a group $\mathbb{G}$ with neutral element $1_\mathbb{G}$, $\mathbb{G}^*$ denotes $\mathbb{G}\backslash\{1_\mathbb{G}\}$. For any two sets $\mathcal{X}$ and $\mathcal{Y}$, denote by $\mathcal{Y}^\mathcal{X}$ the set of functions from $\mathcal{X}$ to $\mathcal{Y}$.

Vectors are denoted in bold font. For two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ in R^n where R is a ring and n a positive integer, $\boldsymbol{a} \circ \boldsymbol{b}$ denotes the Hadamard product of $\boldsymbol{a}$ and $\boldsymbol{b}$, i.e., $\boldsymbol{a} \circ \boldsymbol{b} := \begin{bmatrix} a_1b_1 & \cdots & a_nb_n \end{bmatrix}$.

Group Families. A *group-family generator* G is a PPT algorithm which takes as input a security parameter λ and returns a tuple $(\mathbb{G}, \ell, g)$, with $\mathbb{G}$ a cyclic multiplicative group of prime order ℓ, and $g \in \mathbb{G}$ a generator of $\mathbb{G}$ (i.e. $g \in \mathbb{G}^*$).

Randomness sources and min-entropy. Imperfect randomness is modeled as arbitrary probability distributions with a certain amount of *entropy*. The *min-entropy* notion is used to measure the randomness in such an imperfect random source. A source is said to have k bits of min-entropy if its distribution has the property that each outcome occurs with probability at most 2^{-k}.

Pseudo-Random Functions. A Pseudo-Random Function (PRF) [20] is an efficiently computable function of which the values are computationally indistinguishable from uniformly random values.

Formally, a function $\mathsf{PRF}\colon \mathcal{K}(\lambda) \times \mathcal{X}(\lambda) \to \mathcal{Y}(\lambda)$ is a (T,q,ε)-secure PRF with key space $\mathcal{K}$, input space $\mathcal{X}$ and range $\mathcal{Y}$ (all assumed to be finite) if the advantage

$$\left|\Pr\left[1 \leftarrow \mathcal{A}^{\mathsf{PRF}(K,\cdot)} \colon K \leftarrow_\$ \mathcal{K}\right] - \Pr\left[1 \leftarrow \mathcal{A}^{f(\cdot)} \colon f \leftarrow_\$ \mathcal{Y}^\mathcal{X}\right]\right|$$

of every adversary $\mathcal{A}$ that runs in time at most $T(\lambda)$ is at most $\varepsilon(\lambda)$.

Dodis–Yampolskiy Pseudo-Random Function. Let G be a group family generator. The Dodis–Yampolskiy pseudo-random function [17] in an ℓ-order group $(\mathbb{G}, \ell, g) \leftarrow_\$ \mathsf{G}$ is the map $F\colon (K, x) \in \mathcal{K} \times \mathcal{X} \mapsto g^{1/(K+x)} \in \mathbb{G}^*$, with $\mathcal{K} = \mathbb{Z}_\ell^*$ and $\mathcal{X} \subset \mathbb{Z}_\ell^*$. This PRF is $\left(T/\left(q\lambda^{O(1)}\right), q, \varepsilon q\right)$-secure under the (T, q, ε)-Decisional Diffie-Hellman Inversion (DDHI) assumption [6,7] for G, where $q(\lambda) = O(\log \lambda)$ is an upper-bound on the bit-size of $\mathcal{X}$ for all λ [17, § 4.2].

3 Model

This section formalizes key-generation protocols for arbitrary, predetermined key-generation algorithms. Such a protocol is executed between a *user* $\mathcal{U}$ and a *certification authority* $\mathcal{CA}$. At the end of the protocol, $\mathcal{U}$ obtains a pair of public–secret keys that $\mathcal{CA}$ certifies to be indistinguishable from keys generated by a fixed algorithm KeyGen, and to have been generated with proper randomness. These requirements are formally captured by a model for randomness verifiability given below. The security definition of the model ensures that a protocol satisfying its conditions fulfills the following properties:

1. $\mathcal{CA}$ can infer no more information about the secret key than it would from a public key generated by KeyGen if $\mathcal{U}$'s randomness source has high entropy
2. no external attacker can distinguish a public key generated via the protocol from a public key generation with KeyGen if the randomness source of either $\mathcal{U}$ or $\mathcal{CA}$ has high entropy
3. $\mathcal{U}$ cannot bias the generation of the keys if the randomness source of $\mathcal{CA}$ has high entropy. In particular, $\mathcal{U}$ cannot use the public key as a subliminal channel to convey information.

3.1 Syntax

An interactive asymmetric-key-generation protocol is a triple $\mathsf{IKG} = (\mathsf{Setup}, \mathsf{U}, \mathsf{CA})$ of algorithms such that $\mathsf{Setup}\left(1^\lambda\right) \to pp$ is a probabilistic algorithm which returns public parameters and $\langle \mathsf{U}(pp; r_\mathcal{U}) \rightleftharpoons \mathsf{CA}(pp; r_\mathcal{CA})\rangle \to \langle (pk_\mathcal{U}, sk), pk_\mathcal{CA}\rangle$ are interactive algorithms. At the end of the protocol, the user key-generation algorithm U returns a pair of public–secret keys, and the certificate-authority key-generation algorithm CA returns a public key.

Algorithm Setup may require some randomness, but the parameters it generates can be fixed once for all and used across multi sessions and by several users and authorities. Once parameters are fixed, high-entropy randomness is still needed to securely generate keys, and this is formalized in Section 3.2.

Definition 3.1 (Correctness). *In the O-oracle model, a key-generation protocol IKG is δ-correct w.r.t. a class $\mathscr{A}$ of algorithms if for all $\lambda \in \mathbb{N}$, for every $\mathcal{A} \in \mathscr{A}$,*

$$\Pr\left[pk_\mathcal{U} = pk_\mathcal{CA} \neq \bot\colon \begin{array}{l} pp \leftarrow_\$ \mathsf{Setup}\left(1^\lambda\right) \\ (\mathcal{D}_\mathcal{U}, \mathcal{D}_\mathcal{CA}) \leftarrow_\$ \mathcal{A}^{O(\cdot)}(pp) \\ r_\mathcal{U} \leftarrow_\$ \mathcal{D}_\mathcal{U}, r_\mathcal{CA} \leftarrow_\$ \mathcal{D}_\mathcal{CA} \\ \langle (pk_\mathcal{U}, sk), pk_\mathcal{CA}\rangle \leftarrow \langle \mathsf{U}(pp; r_\mathcal{U}) \rightleftharpoons \mathsf{CA}(pp; r_\mathcal{CA})\rangle \end{array}\right] \geq \delta.$$

Note that the last line of the probability event implicitly implies that U and CA *must terminate.*

The above definition is given in model in which $\mathcal{A}$ has oracle access to $\mathcal{O}$. This latter is used to "distinguish" different models: it may be a random oracle, but it could also simply be an oracle which returns a fixed value (i.e., the common-reference-string model) or no value at all (the standard model). The reason for this distinction is that if a component of the protocol (e.g. a randomized primality-testing algorithm) is not perfectly correct, then its correctness probability is only defined for perfect randomness although the parties only have access to imperfect randomness. However, in the random-oracle model for instance, this imperfect randomness chosen by the algorithm in the definition may depend on the random-oracle queries made by this latter.

3.2 Security

This section gives a game-based security model for key-generation protocols with verifiable randomness. It covers concurrent protocol executions with different instances of protocol algorithms. It is inspired by the BPR model for authenticated key exchange [4] but with key differences.

Protocol Participants. A set of user identities U and a set of certificate-authority identities CA are assumed to be fixed. The union of the those sets form the overall identity space ID. For readability, it is implicitly assumed that during protocol executions, the messages exchanged are always prepended with the instance identifier of the receiving party. Note that several instances of the same algorithm may concurrently run during the game.

Adversaries. The game features a two-stage adversary $(\mathcal{A}_1, \mathcal{A}_2)$. Adversaries $\mathcal{A}_1$ and $\mathcal{A}_2$ may agree on a common strategy before the beginning of the game. That is to say, the strategy may be part of their code, and it may dictate which queries to make (possibly depending on the oracle answers), the order of the queries and so forth. All but the challenge query can only be made by $\mathcal{A}_1$. The role of $\mathcal{A}_2$ is essentially only to guess whether a public key was generated with KeyGen or with the protocol, while $\mathcal{A}_1$ can make arbitrary queries according to the pre-established strategy.

However, $\mathcal{A}_1$ and $\mathcal{A}_2$ cannot communicate after the beginning of the game. It reflects the fact that in practice, an implementer may distribute its key generator, but does not necessarily wiretap the execution of the key-generation protocol for a particular user. From a technical viewpoint, the reason is that in a key-generation protocol, a user has to prove to the authority that she correctly performed her computation. However, the randomness used in these proofs can be used as a subliminal channel to convey information about the secret key. For instance, an engineer could in practice implement a bogus key generator which only terminates the protocol if the first bits of the proof and secret key match. The proof then serves as subliminal channel to leak information about the secret key. Later on, when a user wants to generate a certified public key, if the

$\mathsf{Init}\left(1^\lambda, U, CA, I\right)$

```
$h \leftarrow_\$ \Omega$; $pp \leftarrow_\$ \mathsf{Setup}\left(1^\lambda\right)$
$ID \leftarrow U \cup CA$
for $i \in [\![I]\!]$ and $id \in ID$ do
    $st^i_{id} \leftarrow r^i_{id} \leftarrow \perp$
    $used^i_{id} \leftarrow$ FALSE
    $acc^i_{id} \leftarrow term^i_{id} \leftarrow flag^i_{id} \leftarrow$ FALSE
    $sid^i_{id} \leftarrow pid^i_{id} \leftarrow \perp$
    $sk^i_{id} \leftarrow pk^i_{id} \leftarrow \perp$
$Q_{\mathsf{Reveal}} \leftarrow Q_{\mathsf{Corrupt}} \leftarrow \emptyset$
return $(pin, sin)$
```

$\mathsf{Oracle}(M)$

```
return $h(M)$
```

$\mathsf{Dist}\left(id, i, \mathcal{D}^i_{id}\right)$

```
$r^i_{id} \leftarrow_\$ \mathcal{D}^i_{id}$ // $r^i_{id}$ is simply generated and *not* returned to $\mathcal{A}_1$
```

$\mathsf{Exec}(\mathcal{U}, i, \mathcal{CA}, j)$

```
if $\left(\mathcal{U} \notin U \text{ or } \mathcal{CA} \notin CA \text{ or } used^i_{\mathcal{U}} \text{ or } used^j_{\mathcal{CA}}\right)$ return $\perp$
if $r^i_{\mathcal{U}} \neq \perp$ and $r^j_{\mathcal{CA}} \neq \perp$
    return $\left\langle \mathsf{U}_i\left(pp, r^i_{\mathcal{U}}\right), \mathsf{CA}_j\left(pp, r^j_{\mathcal{CA}}\right)\right\rangle$
return $\perp$ // $\mathcal{A}_1$ must specify distributions beforehand
```

$\mathsf{Send}(id, i, M)$

```
if $r^i_{id} = \perp$ return $\perp$ // $\mathcal{A}_1$ must specify a distribution beforehand
if $term^i_{id}$ return $\perp$
$used^i_{id} \leftarrow$ TRUE
$\left\langle m_{out}, acc, term, sid, pid, pk, sk, st^i_{id}\right\rangle \leftarrow \left\langle \mathsf{IKG}\left(id, st^i_{id}, M; r^i_{id}\right)\right\rangle$
if $acc$ and $\neg acc^i_{id}$
    $sid^i_{id} \leftarrow sid$; $pid^i_{id} \leftarrow pid$
    $acc^i_{id} \leftarrow acc$
if $term$ and $\neg term^i_{id}$ // Set keys only after termination
    $pk^i_{id} \leftarrow pk$; $sk^i_{id} \leftarrow sk$
return $\left(m_{out}, sid, pid, pk, sk, acc, term^i_{id}\right)$
```

$\mathsf{Reveal}(id, i)$

```
$Q_{\mathsf{Reveal}} \leftarrow Q_{\mathsf{Reveal}} \cup \{(id, i)\}$
return $\left(pk^i_{id}, sk^i_{id}\right)$
```

$\mathsf{Corrupt}(id)$

```
$Q_{\mathsf{Corrupt}} \leftarrow Q_{\mathsf{Corrupt}} \cup \{id\}$
for $i \in [\![I]\!]$ $\left\{\text{if } \neg acc^i_{id} \text{ then } flag^i_{id} \leftarrow \text{TRUE}\right\}$
return $\{st^i_{id}\}_{i \in [\![I]\!]}$
```

$\mathsf{Test}_b(id^*, i^*)$

```
if $\left(\exists (id_0, id_1, i, j) \colon pid^i_{id_0} = id_1 \text{ and } pid^j_{id_1} = id_0 \text{ and } acc^i_{id_0}\right.$
   $\left.\text{and } \neg term^j_{id_1}\right)$ return $\perp$
// Once an instance accepts, its partner must eventually terminate
if $\neg term^{i^*}_{id^*}$ return $\perp$
if $flag^{i^*}_{id^*}$ return $\perp$
// Reject if $id^*$ was corrupt before $(id^*, i^*)$ accepted
if $(id^*, i^*) \in Q_{\mathsf{Reveal}}$ or $\left(\exists (id', j) \colon pid^{i^*}_{id^*} = id' \text{ and } pid^j_{id'} = id^*\right.$
        $\left.\text{and } (id', j) \in Q_{\mathsf{Reveal}}\right)$
            return $\perp$
// Reject if the key of $(id^*, i^*)$ or of its partner has been revealed
if $pk^{i^*}_{id^*} \neq \perp$
    if $b = 0$
        $(pk, sk) \leftarrow_\$ \mathsf{KeyGen}\left(1^\lambda\right)$
        return $pk$
    return $pk^{i^*}_{id^*}$
return $\perp$ // Reject if $(id^*, i^*)$ does not have a key
```

Fig. 1. Oracles for the Key-Generation Indistinguishability Experiment.

engineer could wiretap the protocol execution, he could infer some information about her secret key through the proof of correct computation. It is the reason why communication between the adversaries cannot be allowed.

The restriction that $\mathcal{A}_1$ and $\mathcal{A}_2$ cannot communicate after the beginning of the game means that the attacks in which the protocol executions are listened to are excluded, but as explained above, it seems to be a minimal requirement.

Game Overview. At the beginning of the game, the challenger first runs an initialization algorithm. After that, $\mathcal{A}_1$ can make several queries to the algorithm instances. It can in particular

* specify distributions from which randomness is drawn and given as an input to the instances,
* ask for the protocol to be executed between different instances of the protocol algorithms without its intervention, i.e., perform passive attacks,
* perform active attacks by sending messages to algorithm instances,
* later on reveal the keys that were generated by a particular instance,
* corrupt a party (user or certificate authority), and thereby gain access to the state of all its algorithm instances.

As for $\mathcal{A}_2$, it can reveal keys or make a test query that returns either (with probability 1/2 each) keys freshly generated by the key-generation algorithm or keys generated by instances of its choice via queries made by $\mathcal{A}_1$. Adversary $\mathcal{A}_2$ must eventually return a guess for the origin of the keys it was returned, and $(\mathcal{A}_1, \mathcal{A}_2)$ wins the game if the guess of $\mathcal{A}_2$ is correct.

Initialization & Game Variables. During the initialization phase, game variables are declared for every instance of the protocol algorithms. Assume that there are at most $I = I(\lambda)$ instances of any participant id. Each instance $i \in I$ of a participant id maintains a state st_{id}^{i}. A session identity sid_{id}^{i} and a partner identity pid_{id}^{i} allow to match instances together in protocol executions. It is assumed that for each sid_{id}^{i} there can be at most one partner instance, i.e., one pair (id', j) such that $pid_{id}^{i} = id'$ and $sid_{id}^{i} \coloneqq \left(id, i, id', j, {sid_{id}^{i}}'\right)$.

Public/secret-key variables (denoted pk_{id}^{i} and sk_{id}^{i}) hold the keys that were output, if any, by the ith instance of the algorithm of party id at that step of the computation. For certificate authorities, the secret keys are always set to $\perp$.

A variable $used_{id}^{i}$ indicates whether the adversary has performed an active attack on the ith algorithm instance of participant id.

Variables acc_{id}^{i} and $term_{id}^{i}$ respectively indicate whether the algorithm of the ith instance of participant id has accepted and terminated. As in the BPR model [4], termination and acceptance are distinguished. When an instance terminates, it does not output any further message. However, it may accept at a certain point of the computation, and terminate later. In the present context, it may for instance occur when an instance expects no further random input from its partner instance, and the rest of its computation is purely deterministic. It may then only terminate after finishing its computation. This distinction is crucial for the security definition. It is important to exclude the trivial case in which, although every computation was honestly performed, a user discards the public key if it does not follow a certain pattern, thereby influencing the distribution of the output public key (i.e., perform rejection sampling), and possibly using it as subliminal channel to convey information about the secret key.

Another variable flag_{id}^{i} (new compared to the BPR model) indicates whether party id was corrupted before its ith instance had accepted. Recall that acceptance intuitively means that an instance expects no further random input from its partner instance. As long as flag_{id}^{i} is set to FALSE, the only information the adversary has about r_{id}^{i} is its distribution and therefore, if this distribution has high min-entropy, the adversary cannot bias the generation of the keys.

A variable r_{id}^{i} holds the random string to be used the ith instance of the algorithm of id.

The challenger maintains a set (initially empty) Q_{Reveal} of identity–instance pairs of which the keys were revealed. It also maintains a set (initially empty) Q_{Corrupt} of corrupt identities.

At the end of the initialization phase, the public parameters, the sets of participants and the user public keys are returned in a public input pin, and the rest is set in a secret input sin. That is, $pin \leftarrow (pp, U, CA, I, (pk_{id})_{id})$ and $sin \leftarrow \Big(pin, (sk_{id})_{id}, \big(st_{id}^{i}, sid_{id}^{i}, pid_{id}^{i}, acc_{id}^{i}, term_{id}^{i}, used_{id}^{i}\big)_{i,id},\ Q_{\mathsf{Corrupt}},\ Q_{\mathsf{Reveal}}\Big)$. The secret input sin is later made available to all oracles.

Oracles. Throughout the game, adversary $\mathcal{A}_1$ is given access to the oracles summarized below and defined in Figure 1. It can query them *one at a time.*

* Oracle : gives access to a function h chosen uniformly at random from a probability space Ω. The adversary and the protocol may depend on h. The probability space Ω specifies the model in which the protocol is considered. If it is empty, then it is the standard model. If it is a space of random functions, then it is the random oracle model. As for the Common-Reference String (CRS) model, Ω is a space of constant functions.
* Dist : via this oracle, the adversary specifies the distribution $\mathcal{D}_{id}^{i}$ from which the randomness of the ith instance of id is drawn. These distributions are always assumed to be independent of oracle Oracle. However, the distributions specified by the adversary for different instances *can be correlated in any way.* Oracle Dist then generates a bit string r_{id}^{i} according to the input distribution and does *not* return it to the adversary. Whenever oracle Exec or oracle Send is queried on (id, i), it uses randomness r_{id}^{i} for its computation. This new (compared to the BPR model) oracle is essential to express requirements on the minimal entropy used by the instances, and also to express reasonable winning conditions. It allows to properly capture properties like the fact that (1) the authority cannot infer any information about the secret key if the randomness of the user algorithm has high entropy, (2) that the output keys are indistinguishable from keys generated with the key-generation algorithm if the randomness used by the algorithm of either of the parties has high entropy, or (3) that a potentially malicious user algorithm cannot

bias the distribution of the output keys if the randomness of the authority algorithm has high entropy. That is first because the test query later made by $\mathcal{A}_2$ requires the min-entropy of the randomness of either the challenge instance or of its partner to be high. It is also due to the fact that the adversary cannot corrupt the challenge instance (thus learning its randomness) before the partner randomness necessary to generate the challenge key is committed, which is monitored by the flags. It for instance means that if the CA is the target of the test and the adversary plays the role of a user algorithm (in which case the partner randomness is considered to have nil entropy) and possibly deviates from the protocol, then the test CA must be given high-entropy randomness and the definition ensures that the resulting key is indistinguishable from keys generated with KeyGen.

* Exec : returns the transcript of an honest (i.e., without the interference of the adversary) protocol execution between the ith instance of U and the jth instance of CA. The protocol is executed with the random strings generated for these instances by oracle Dist on the input of adversarial distributions. The notations U_i and CA_j mean that algorithms U and CA are executed using the state of the ith instance of U and the jth instance of CA respectively. It is implicitly assumed that the states $acc^i_{\mathcal{U}}$, $term^i_{\mathcal{U}}$, $acc^j_{\mathcal{CA}}$ and $term^j_{\mathcal{CA}}$ are set to TRUE after an honest protocol execution. Moreover, if the termination variable of either party is set to TRUE, the protocol is not executed and $\perp$ is returned. In essence, by querying oracle Exec, adversary $\mathcal{A}_1$ performs a passive eavesdropping attack.
* Send : adversary $\mathcal{A}_1$ can perform active attacks via this oracle. $\mathcal{A}_1$ can send any message to an instance of its choice, e.g., the ith instance of a user algorithm, which runs the honest protocol algorithm of the corresponding party on the input of the message chosen by the adversary.
  To prompt the ith instance of id to initiate a protocol execution with the jth instance of id', adversary $\mathcal{A}_1$ can make a Send query on $(id, i, (id', j))$. $\mathsf{IKG}(id, *)$ denotes the IKG algorithm of party id, i.e., either U or CA. The algorithm is executed using the randomness generated by oracle Dist for that instance. (Note that the input random string may be used only at certain steps of the computation.) The oracle then returns the output of the instance to the adversary. It also specifies if this instance accepted and/or terminated, and returns the session identifier and the identity of its partner in the protocol execution, as well as the public and secret keys returned by this instance, if any. Note that if the instance is that of a certificate-authority algorithm, the secret key is always set to $\perp$.
* Reveal : on input (id, i), returns the keys held by the ith instance of the algorithm of id. The couple (id, i) is added to the set Q_{Reveal} of revealed keys.
* $\mathsf{Corrupt}$: on input id, returns the states of all the instances of the algorithm of id. The identity id is added to the set Q_{Corrupt} of corrupt identities. Besides, for any instance i of id, if it has not yet accepted, $flag^i_{id}$ is set to TRUE.

Remark 3.1. The first main difference with the BPR model is the new oracle Dist. It allows to capture an adversary running several instances of the protocol with correlated randomness. In the new model, it is also important to express winning conditions that exclude the trivial (and unavoidable) rejection-sampling attack. Another difference is that the variable flag_{id}^{i} is set to TRUE if $\mathcal{A}_1$ corrupts id before its ith instance has accepted. It is to say that for instance, if an adversary (e.g., a malicious user algorithm) knows the randomness of the other party (by corrupting the CA) before it has "committed" to its randomness, then that party can influence the resulting key and break property (3).

As for adversary $\mathcal{A}_2$, it is given access to oracles Oracle, Reveal and to oracle

* Test_b : on input (id^*, i^*), it returns the public key $pk_{id^*}^{i^*}$ generated via IKG (with an Exec query or Send queries) if $b = 0$ or a fresh public key generated via KeyGen if $b = 1$.
  An important restriction on this query is that the following condition must be satisfied: for any instance i of the algorithm of a party id_0, once it has accepted, i.e., once $acc_{id_0}^{i}$ is set to TRUE, the partner instance algorithm, say the jth instance of id_1, must eventually terminate, i.e., $term_{id_1}^{j}$ must have been set to TRUE as well by the time of query Test. It prevents $\mathcal{A}_1$ from biasing the distribution of the keys by prematurely aborting the protocol although it was followed, if the resulting key does not follow a certain pattern, and which would allow $\mathcal{A}_2$ to guess b with a non-negligible advantage.
  The other restrictions are simply that i^*-th instance of id^* must have terminated, that id^* was not corrupt before (id^*, i^*) had accepted[6], that neither the key of the i^*th instance of id^* nor of its partner instance has been revealed, and that the i^*th instance of id^* must already hold a key.
  Note that $\mathcal{A}_2$ can query Test only once. A definition with multiple queries would be asymptotically equivalent via a standard hybrid argument.

Adversary $\mathcal{A}_2$ must eventually return a bit b' as a guess for the origin (i.e., either IKG or KeyGen) of the key returned by oracle Test_b.

To achieve any form of indistinguishability from a key-generation algorithm, it is clear that either the distribution $\mathcal{D}_{id^*}^{i^*}$ or the distributions $\mathcal{D}_{id'}^{j}$ for the partner instance (j, id') of (i^*, id^*) must have high entropy. Indeed, if distributions with low entropy were allowed, $\mathcal{A}_1$ and $\mathcal{A}_2$ could agree on these identities, instances and distributions beforehand. Adversary $\mathcal{A}_2$ could then simply return 1 if and only if the challenge key is the most likely key w.r.t. $\mathcal{D}_{id^*}^{i^*}$ and $\mathcal{D}_{id'}^{j}$, and thereby win the game with a non-negligible advantage.

[6] To understand why it is necessary for id^* not to be corrupt before (id^*, i^*) accepts even though $\mathcal{A}_1$ and $\mathcal{A}_2$ do not communicate, suppose that this condition were not imposed and consider the following strategy which allows $(\mathcal{A}_1, \mathcal{A}_2)$ to trivially win: $\mathcal{A}_1$ and $\mathcal{A}_2$ agree on (id^*, i^*) and on a distribution $\mathcal{D}_{id^*}^{i^*}$. Adversary $\mathcal{A}_1$ prompts (id^*, i^*) to initiate a protocol execution by making a Send query. It then corrupts id^* and obtains $st_{id^*}^{i^*}$, from which it can read $r_{id^*}^{i^*}$. Adversary $\mathcal{A}_1$ could then play the role of its partner and adapt the messages it sends to make sure that the resulting public key follows a certain pattern known to $\mathcal{A}_2$. This latter would then be able to win the game with a non-negligible advantage.

A parameter κ for the maximal min-entropy of $\mathcal{D}_{id^*}^{i^*}$ and $\mathcal{D}_{id'}^{j}$ specified by $\mathcal{A}_1$ is therefore introduced. If the adversary modified any message from the partner (j, id') of (id^*, i^*) before (id^*, i^*) accepts, then $\mathcal{D}_{id'}^{j}$ is set to be the Dirac mass at the zero bit-string by convention (and it thus has no entropy). The underlying idea is that as long as at least one of the two parties has a randomness source with high entropy, the key returned at the end of the protocol should be indistinguishable from a key generated by the KeyGen algorithm. The security of a key-generation protocol is then defined for adversaries that specify challenge distributions with min-entropy at least κ.

Definition 3.2 (Indistinguishability). *An interactive key-generation protocol* IKG *is* $(T, q_{\mathsf{Oracle}}, q_{\mathsf{Dist}}, q_{\mathsf{Exec}}, q_{\mathsf{Send}}, q_{\mathsf{Reveal}}, q_{\mathsf{Corrupt}}, \kappa, \varepsilon)$*-indistinguishable from a key-generation algorithm* KeyGen *(running on uniform randomness) if for all* $\lambda \in \mathbb{N}$*, for every adversary* $(\mathcal{A}_1, \mathcal{A}_2)$ *that runs in time at most* $T(\lambda)$ *and makes at most* q_O *queries to* $O \in \{\mathsf{Oracle}, \mathsf{Dist}, \mathsf{Exec}, \mathsf{Send}, \mathsf{Reveal}, \mathsf{Corrupt}\}$*, and such that* $\max\left(H_\infty\left(\mathcal{D}_{id^*}^{i^*}\right), H_\infty\left(\mathcal{D}_{id'}^{j}\right)\right) \geq \kappa$ *for query* Test*, the advantage (function of* λ*)*

$$\left|\Pr\left[b = b' : \begin{array}{l} (pin, sin) \leftarrow \mathsf{Init}\left(1^\lambda, U, CA, I\right) \\ O_1 \leftarrow \{\mathsf{Oracle}, \mathsf{Dist}, \mathsf{Exec}, \mathsf{Send}, \mathsf{Reveal}, \mathsf{Corrupt}\} \\ \mathcal{A}_1^{O_1(sin,\cdot)}(pin) \\ b \leftarrow_\$ \{0,1\} \\ O_2 \leftarrow \{\mathsf{Oracle}, \mathsf{Reveal}, \mathsf{Test}_b\} \\ b' \leftarrow \mathcal{A}_2^{O_2(sin,\cdot)}(pin) \\ \text{return } (b, b') \end{array}\right] - 1/2\right|$$

of $(\mathcal{A}_1, \mathcal{A}_2)$ *is at most* $\varepsilon(\lambda)$.

From a practical perspective, this definition (which implies requirement 3 as it enforces indistinguishability from keys generated by IKG) means that keys generated via a protocol satisfying the definition above are not subject to randomness vulnerabilities such as the ROCA vulnerabilities [25] and those [23,21] in which several public keys are generated with correlated randomness sources.

4 Generic Constructions

This section presents a protocol that covers a wide class of key-generation algorithms, namely those that can be represented as probabilistic circuits, and another protocol specific to the generation of RSA keys. The first protocol is of theoretical interest and shows that randomness verifiability can be achieved for wide class of key-generation algorithms, whereas the second protocol is a solution that can actually be used in practice.

4.1 Key-Generation Protocol with Verifiable Randomness for Probabilistic Circuits

This section gives a key-generation protocol with verifiable randomness for a large class of key-generation algorithms. The focus is here on the class of key-generation algorithms that can be modeled as *probabilistic circuits*.

The advantage of probabilistic circuits compared to general Turing Machines for this purpose is that the running time of a probabilistic circuit is independent of the random inputs. In a key-generation protocol with verifiable randomness, the user has to prove to the authority that she correctly performed her computation. Having a constant running time then ensures that no one can infer any information about the secret key from the statement proved by the user or the proof itself. It prevents malicious user algorithms from using the proofs as subliminal channels to pass information about the secret key.

To understand why it is important for the running time to be constant, consider the following artificial random number generator. To generate a k-bit string $t = (t_0, \ldots, t_{k-1})$, it consists in flipping a random coin s several times for each bit t_i and to set this bit to the parity of the number of flipped coins to obtain the first "Head". It produces a k-bit string uniformly distributed within expected time complexity $O(k)$ and it could be used as a secret-key generation algorithm (and the public key would then be a deterministic function of the generated secret key). See the full version [6] for a formal description of the algorithm. For a user to prove that she correctly generated the random bit string t, she would have to commit to the t_i values and compute a proof on the successive s values. However, each t_i is simply the parity of the number of trials before $s = 0$. Therefore, from the number of s values for which the user has to perform a proof, the authority can infer t_i. For example, if the user generated two s values for t_1, the authority knows that $t_1 = 0$.

In other words, the statement of the proof itself reveals some information about the secret key to the certification authority; and the issue is here that the running time changes from one random run of the algorithm to the other. Restricting to probabilistic circuits eliminates this issue.

The restriction to circuits comes at a cost though. It for instance excludes the class of algorithms for which there is no known circuit that can represent them. It is for instance the case of algorithms that must efficiently generate primes during the process. Indeed, there is no known circuit that can efficiently generate prime numbers. On this ground, the generic protocol for probabilistic circuits of Section 4.1 does not apply to the RSA-key generation for instance[7]. See rather Section 4.2 for the specific case of RSA-key generation with verifiable randomness for arbitrary properties that the keys must satisfy.

Before describing our protocol, we first formally define probabilistic circuits.

Probabilistic Circuits. A probabilistic circuit is essentially a deterministic circuit augmented with uniformly random gates. The random gates produce independent and uniform random bits that are sent along their output wires.

We equivalently define a probabilistic circuit as a uniform random variable over a finite collection of deterministic boolean circuits. These boolean circuits

[7] One can construct families of probabilistic "circuits" which output an RSA key but *only* with overwhelming probability (and not probability 1) by relying on the prime number theorem and Chernoff's bound. However, such constructions would have large gate complexity and randomness complexity and applying our generic construction to such circuits family would result in schemes with prohibitive efficiency.

are restricted to have the same amount n of input variables, and r fixed inputs. The number r of fixed inputs depends on the security parameter 1^λ. Denote such a circuit as $\Gamma_{b_1 \cdots b_r}(x_1, \ldots, x_n)$, with $x_1, \ldots, x_n$ the input variables and $b_1, \ldots, b_r$ the fixed inputs. To each element in $\{0,1\}^r$ corresponds a circuit in the collection with the bit string as fixed inputs, so that there are 2^r circuits in the collection. However, these circuits are not required to form a uniform family (i.e., they are not required to be output by a single Turing machine); the circuit families here considered can be non-uniform.

A probabilistic circuit Γ is then defined as a uniform random variable over the set (of circuits) $\{\Gamma_b\}_{b \in \{0,1\}^r}$. Namely, for input variables $x_1, \ldots, x_n$, the evaluation $\Gamma(x_1, \ldots, x_n)$ is a uniform random variable over the set (of values) $\{\Gamma_b(x_1, \ldots, x_n)\}_{b \in \{0,1\}^r}$. If $\omega \in \{0,1\}^r$ denotes the random input to the probabilistic circuit Γ, the evaluation $\Gamma(x_1, \ldots, x_n; \omega)$ is then $\Gamma_\omega(x_1, \ldots, x_n)$.

The advantage of this second definition is that randomness is invoked only once instead of invoking it for each of the r random gates. To generate keys, PRFs are often used to provide random bit strings from small secret seeds. As the goal is to build a key-generation protocol which allows the CA to certify that the keys are generated with high-entropy randomness, the user will have to prove that she correctly performed the pseudo-random evaluations. Invoking randomness only once then allows to invoke the PRF only once in the protocol.

Generic Protocol. We now give a two-party protocol in the CRS model to generate, with verifiable randomness, keys computed by probabilistic circuits. Requiring that keys are generated with verifiable randomness here means that the random inputs to the circuits must be uniformly generated in a verifiable manner. The deterministic inputs can simply be considered as public parameters.

Building Blocks. The protocol involves (see the full version [6] for definitions)

– a function family $\mathcal{H} = \{H_{hk}\}_{hk \in \{0,1\}^{d(\lambda)}}$ which is a universal computational extractor w.r.t. unpredictable sources
– a two-source extractor Ext with key space $\{0,1\}^{\delta(\lambda)}$
– an extractable commitment scheme $\mathscr{C} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{ComVf}, \mathsf{TSetup}, \mathsf{ExtCom})$ for the user algorithm to commit to its random string *before receiving any input from the CA*, thereby preventing it from biasing the distribution of the keys. The parameters returned by Setup are implicit inputs to the other algorithms of $\mathscr{C}$
– a non-interactive, extractable, zero-knowledge proof system Π with $\Pi = \left(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verf}, \mathsf{TSetup}^{zk}, \mathsf{Sim}, \mathsf{TSetup}^{ext}, \mathsf{Ext}\right)$ for the relation

$$\mathcal{R}_\Pi \coloneqq \Big\{\Big((x_i)_i, k, C, r_{\mathcal{CA}}, pk; r'_{\mathcal{U}}, d, sk\Big) : \mathsf{ComVf}\left(C, r'_{\mathcal{U}}, d\right) = 1 \\ \wedge (pk, sk) = \Gamma\left(x_1, \ldots, x_n; \mathsf{Ext}_k\left(r'_{\mathcal{U}}, r_{\mathcal{CA}}\right)\right)\Big\},$$

– a pseudo-random function PRF to generate the randomness for $\Pi.\mathsf{Prove}$.

Parameters. Given a circuit Γ with deterministic inputs $x_1, \ldots, x_n$, to generate public parameters for the protocol on the input a security parameter 1^λ, run $pp_{\mathscr{C}} \leftarrow \mathscr{C}.\mathsf{Setup}\left(1^\lambda\right)$ ($pp_{\mathscr{C}}$ is a tacit input to the algorithms of $\mathscr{C}$), $crs \leftarrow \Pi.\mathsf{Setup}\left(1^\lambda\right)$, and generate $hk \leftarrow_\$ \{0,1\}^{d(\lambda)}$ and $k \leftarrow_\$ \{0,1\}^{\delta(\lambda)}$. Return $pp \leftarrow (crs, pp_{\mathscr{C}}, hk, k, x_1, \ldots, x_n)$.

Formal Description. Consider the interactive protocol IKG_Γ on Figure 2 between a user $\mathcal{U}$ and a certification authority $\mathcal{CA}$. Each algorithm maintains acceptance and termination variables acc_{id} and $term_{id}$, for $id \in \{\mathcal{U}, \mathcal{CA}\}$, initially set to FALSE. On the input of pp and of their respective random strings $r_{\mathcal{U}}$ and $r_{\mathcal{CA}}$, the party algorithms proceed as follows:

1. U separates the domain of H_{hk} in two and applies it to its randomness. It commits to the first output with the second output as randomness, and sends the resulting commitment C to $\mathcal{CA}$
2. CA, upon receiving the commitment from $\mathcal{U}$, sets $acc_{\mathcal{CA}} \leftarrow$ TRUE and sends its random string $r_{\mathcal{CA}}$ to $\mathcal{U}$
3. U, upon receiving $r_{\mathcal{CA}}$ from $\mathcal{CA}$, sets $acc_{\mathcal{U}} \leftarrow$ TRUE. Next, it extracts a seed s with Ext from the joint randomness. It evaluates Γ on $x_1, \ldots, x_n$ and s, and obtains a key pair (pk, sk). It generates another seed s' with H_{hk}. Algorithm U then evaluates PRF mode on s' to generate the randomness necessary to compute $\Pi.\mathsf{Prove}$ since U has no other random string than $r_{\mathcal{U}}$ available, i.e., it computes $r_\Pi \leftarrow \mathsf{PRF}(s', 0)$. Algorithm U then proves that it followed the protocol and correctly evaluated Γ at $x_1, \ldots, x_n$, i.e., it computes a proof $\pi \leftarrow \Pi.\mathsf{Prove}\left(crs, ((x_i)_i, k, C, r_{\mathcal{CA}}, pk), \left(r'_{\mathcal{U}}, d, sk\right); r_\Pi\right)$. After that, it erases all variables but pk, sk, π, sends pk and π to $\mathcal{CA}$, returns (pk, sk) and sets $term_{\mathcal{U}} \leftarrow$ TRUE
4. CA, upon receiving (pk, π) from $\mathcal{U}$, verifies the proof. If the proof is valid, it returns pk, otherwise it returns $\perp$. It then sets $term_{\mathcal{CA}} \leftarrow$ TRUE.

Correctness & Indistinguishability. In the full version [6], we show that IKG_Γ is 1-correct w.r.t. all algorithms if $\mathscr{C}$ is correct and if Π is complete. Moreover, it is indistinguishable from Γ in the CRS model for sources with min-entropy at least $\kappa = \max(\kappa_{\mathcal{H}}, \kappa_{\mathsf{Ext}})$ if Ext is a $(\kappa_{\mathsf{Ext}}, \varepsilon_{\mathsf{Ext}})$-extractor for $\kappa_{\mathsf{Ext}} \leq \min(|r'_{\mathcal{U}}|, |r_{\mathcal{CA}}|)$, if $\mathcal{H}$ is UCE-secure w.r.t. simply unpredictable sources of min-entropy at least $\kappa_{\mathcal{H}}$, if $\mathscr{C}$ extractable and hiding, if Π is extractable and composable zero-knowledge, and if PRF is a secure PRF.

Discrete-Logarithm Keys. The full version of the paper [6] presents a simple illustration of this generic protocol (but rather in the random-oracle model for better efficiency) applied to discrete-logarithm keys.

4.2 RSA-Key Generation Protocol with Verifiable Randomness

This section gives a two-party protocol for RSA-key generation with verifiable randomness between a user $\mathcal{U}$ and a certification authority $\mathcal{CA}$. The resulting

$\mathsf{U}\left(crs, pp_{\mathscr{C}}, hk, k, x_1, \ldots, x_n; r_{\mathcal{U}}\right)$		$\mathsf{CA}\left(crs, pp_{\mathscr{C}}, hk, k, x_1, \ldots, x_n; r_{C\mathcal{A}}\right)$
$r'_{\mathcal{U}} \leftarrow H_{hk}\left(0 \\| r_{\mathcal{U}}, 1^{\lvert r'_{\mathcal{U}} \rvert}\right)$		
$\rho_{\mathcal{U}} \leftarrow H_{hk}\left(1 \\| r_{\mathcal{U}}, 1^{\lvert \rho_{\mathcal{U}} \rvert}\right)$		
$(C, d) \leftarrow \mathsf{Com}\left(r'_{\mathcal{U}}; \rho_U\right)$		
	$\xrightarrow{C}$	
	$\xleftarrow{r_{C\mathcal{A}}}$	
$s \leftarrow \mathsf{Ext}_k\left(r'_{\mathcal{U}}, r_{C\mathcal{A}}\right)$		
$(pk, sk) \leftarrow \Gamma(x_1, \ldots, x_n; s)$		
$s' \leftarrow H_{hk}\left(2 \\| r_{\mathcal{U}}, 1^{\lvert s' \rvert}\right)$		
$\pi \leftarrow \Pi$ proof of correct computation with random string $\mathsf{PRF}\left(s', 0\right)$		
Erase all variables but pk, sk, π	$\xrightarrow{pk, \pi}$	$\Pi.\mathsf{Verf}\left(crs, \left((x_i)_i, k, C, r_{C\mathcal{A}}, pk\right), \pi\right) \overset{?}{=} 1$
return (pk, sk)		return pk

Fig. 2. Key-Generation Protocol with Verifiable Randomness for Probabilistic Circuits.

keys can be used in any RSA cryptosystem. The protocol attests that the resulting keys were generated with high-entropy randomness and that they satisfy (fixed) arbitrary properties. These properties are captured by a relation

$$\mathcal{R}_W \coloneqq \{(N, e \in \mathbb{Z}; p, q \in W \subseteq \mathbb{P}) \colon p \neq q \wedge N = pq \wedge \gcd(e, \varphi(N)) = 1\}$$

to which the keys generated should belong, where W is a set that defines the predicates p and q must satisfy, e.g., $p = q = 3 \mod 4$ or p and q are safe primes. Its relative language is denoted $\mathcal{R}_W$. Efficient proof systems for such properties exist [31,11,1], though none of them aims at proving that the keys were generated with proper randomness.

In comparison, the protocol by Juels and Guajardo [22] only guarantees the first two properties, and does *not* ensure that the user algorithm cannot bias the distribution of the keys. Without the third property, an interactive key-generation protocol is only beneficial if the user does not have high-entropy randomness locally whereas the CA does, otherwise it is only a burden for the user. On the other hand, the third property additionally guarantees the end user that if the CA has high-entropy randomness, her keys are not faulty.

As for the attestation scheme of Benhamouda et al. [5], it allows to prove that the RSA primes were generated with an arbitrary generator; and the protocols of Camenisch and Michels [11], of Auerbach and Poettering [1], and of Goldberg et al. [19], only allow to prove that RSA primes satisfy certain properties, not that they were generated with high entropy. In a sense, our goal is complementary to that of proving that RSA moduli satisfy certain properties without proving that the keys were generated with high-entropy randomness.

RSA Key-Generation Algorithm. The NIST standard [26] for the RSA [28] key-generation algorithm, further denoted $\mathsf{KeyGen}_{\mathrm{RSA}}$, is the following:

– choose at random two distinct large primes p and q
– compute $N \leftarrow pq$ and $\varphi(N) \leftarrow (p-1)(q-1)$
– choose an integer $2^{16} < e < 2^{256}$ such that $\gcd(e, \varphi(N)) = 1$ (e may be chosen deterministically or at random); compute $d \leftarrow e^{-1} \mod \varphi(N)$
– Return $pk \leftarrow (N, e)$ and $sk \leftarrow (N, d)$.

Equivalently, the secret key sk can be set to (p, q, e) instead of (N, d) as one can compute (N, d) from (p, q, e) and vice-versa. It is this variant that is hereafter considered. To formally capture the requirement on p and q to be large, a parameter $b = b(\lambda)$ that specifies the bit-length of p and q is introduced.

Interpretation. There is some ambiguity as to how p and q are generated. The interpretation (which follows how the algorithm would implemented in practice) of $\mathsf{KeyGen}_{\mathrm{RSA}}$ in the rest of the paper is first that there exists a PPT primality-test algorithm $\mathsf{PrimeTest}_W(\lambda, b, e, p) \to \zeta \in \{0, 1\}$ (parameter λ is further omitted from its syntax) which tests whether an integer p is in W, b-bit long and such that $\gcd(e, (p-1)) = 1$. Algorithm $\mathsf{KeyGen}_{\mathrm{RSA}}$ then generates, uniformly at random, integers in $[\![2^{b-1}, 2^b - 1]\!]$ until it finds an integer p such that $\mathsf{PrimeTest}_W(b, e, p) = 1$, and continues until it finds a second one $q \neq p$ such that $\mathsf{PrimeTest}_W(b, e, q) = 1$. If no such two integers are found in a specified number of iterations $T_{\mathrm{RSA}}(\lambda)$, the algorithm aborts and returns an invalid pair, e.g., $(0, 0)$. The random variable with values in $\{0, 1, 2\}$ that counts the number of distinct primes found in at most $T_{\mathrm{RSA}}(\lambda)$ iterations is further denoted ctr_{RSA}.

Protocol. We now describe our protocol, further denoted $\mathsf{IKG}_{\mathrm{RSA}}$, to generate RSA keys with verifiable randomness. The protocol is given in the random-oracle model to allow for practical efficiency.

Building Blocks. The protocol builds on

– the same primality-test algorithm $\mathsf{PrimeTest}_W$ as the one run by $\mathsf{KeyGen}_{\mathrm{RSA}}$. It is said to be *$\delta$-correct* if with probability at most $1-\delta$, $\mathsf{PrimeTest}_W(b, e, p) = 0$ for $p \in W \cap [\![2^{b-1}, 2^b - 1]\!]$ such that $\gcd(e, (p-1)) = 1$, or $\mathsf{PrimeTest}_W(b, e, p) = 1$ for $p \notin W \cap [\![2^{b-1}, 2^b - 1]\!]$ or such that $\gcd(e, (p-1)) > 1$ (i.e., it is an upper-bound on the probability that it returns a false negative or a false positive)
– a random oracle of which the domain is separated to obtain pairwise independent random oracles $\mathcal{H}$, $\mathcal{H}_C$, $\mathcal{H}_\Pi$ and $\mathcal{H}_{\Pi_W}$
– a commitment scheme $\mathscr{C} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{ComVf})$ for the user algorithm to commit to its random string *before* receiving any input from the CA. The parameters returned by Setup are tacit inputs to $\mathscr{C}$ other algorithms.
– a pseudo-random function PRF with range (non-empty) $R_{\mathsf{PRF}} \subseteq \mathbb{N}$ for $\boldsymbol{\mathcal{U}}$ to generate the RSA primes from the seed extracted with $\mathcal{H}$
– an extractable non-interactive zero-knowledge (NIZK) argument system $\Pi_C = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verf}, \mathsf{Sim}, \mathsf{Ext})$ for the relation $\left\{\left(C; r'_{\mathcal{U}}, d\right) : \mathsf{ComVf}(C, r'_{\mathcal{U}}, d) = 1\right\}$ with random oracle $\mathcal{H}_C$, i.e., for the user to prove knowledge of an opening to her committed randomness

- an extractable NIZK argument system $\Pi = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verf}, \mathsf{Sim}, \mathsf{Ext})$ with random oracle $\mathcal{H}_\Pi$ for the relation

  $$\Big\{\Big(C, r_{\mathcal{CA}}, N, (a_\gamma)_{\gamma\neq i,j}; r'_{\mathcal{U}}, d, a_i, a_j\Big) : \mathsf{ComVf}\left(C, r'_{\mathcal{U}}, d\right) = 1,\ s = r'_{\mathcal{U}} \oplus \mathcal{H}(r_{\mathcal{CA}}),$$
  $$\forall\gamma \in [\![j]\!]\ a_\gamma = \mathsf{PRF}(s,\gamma),\ 2^{b(\lambda)-1} \leq a_i, a_j \leq 2^{b(\lambda)} - 1, N = a_i a_j \text{ in } \mathbb{N}\Big\},$$

  i.e., for the user to prove the RSA primes are really the *first two primes* generated with the seed derived from the committed randomness and the randomness of the CA. This relation is further denoted $\mathcal{R}_\Pi$
- a NIZK argument system $\Pi_W = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verf}, \mathsf{Sim})$ with random oracle $\mathcal{H}_{\Pi_W}$ for relation $\mathcal{R}_W$
- another pseudo-random function PRF' with variable output length (encoding in unary as last input) for $\mathcal{U}$ to generate the randomness necessaryto compute $\Pi_C.\mathsf{Prove}$, $\mathsf{PrimeTest}_W$, $\Pi.\mathsf{Prove}$ and $\Pi_W.\mathsf{Prove}$, as the only available randomness to the parties are their input random bit strings.

Throughout the protocol, e is assumed (without loss of generality) to be a fixed[8], hard-coded value in U. For the sake of simplicity, e is further assumed to be prime, e.g., $e = 65537$ (it is a value commonly used in practice).

Parameters. Given a security parameter 1^λ and a function $T\colon \mathbb{N} \to \mathbb{N}_{>1}$ that gives an upper bound on the number of iterations in Algorithm 1 (and thus the running time of U), to generate parameters for $\mathsf{IKG}_{\mathrm{RSA}}$, run $pp_{\mathscr{C}} \leftarrow \mathscr{C}.\mathsf{Setup}\left(1^\lambda\right)$, $pp_{\Pi_C} \leftarrow \Pi_C.\mathsf{Setup}\left(1^\lambda\right)$, $pp_\Pi \leftarrow \Pi.\mathsf{Setup}\left(1^\lambda\right)$ and $pp_{\Pi_W} \leftarrow \Pi_W.\mathsf{Setup}\left(1^\lambda\right)$. Set and return $pp \leftarrow \left(b(\lambda), T(\lambda), pp_{\mathscr{C}}, pp_{\Pi_C}, pp_\Pi, pp_{\Pi_W}\right)$.

Formal Description. Consider the interactive protocol on Figure 3 between a user $\mathcal{U}$ and a certification authority $\mathcal{CA}$. Each algorithm maintains acceptance and termination variables acc_{id} and $term_{id}$ for $id \in \{\mathcal{U}, \mathcal{CA}\}$ initially set to FALSE. The party algorithms proceed as follows:

1. U applies the random oracle $\mathcal{H}$ twice to its randomness $r_{\mathcal{U}}$ to compute $r'_{\mathcal{U}} \leftarrow \mathcal{H}(0\|r_{\mathcal{U}})$ and $\rho_U \leftarrow \mathcal{H}(1\|r_{\mathcal{U}})$, commits to $r'_{\mathcal{U}}$ with ρ_U as random string. Next, a seed $s' \leftarrow \mathcal{H}(2\|r_{\mathcal{U}})$ from which it derives the randomness necessary to compute $\Pi_C.\mathsf{Prove}$, and computes of proof of knowledge of an opening to the commitment. U sends the commitment and the proof to $\mathcal{CA}$
2. CA, upon receiving the commitment and the proof from $\mathcal{U}$, sets $acc_{\mathcal{CA}} \leftarrow$ TRUE. It verifies the proof and if it holds, sends its randomness to $\mathcal{U}$, and otherwise returns $\perp$ and sets $term_{\mathcal{CA}} \leftarrow$ TRUE
3. U, upon receiving $r_{\mathcal{CA}}$ from $\mathcal{CA}$, sets $acc_{\mathcal{U}} \leftarrow$ TRUE. It extracts a seed s with $\mathcal{H}$ from the joint randomness. It continues by generating by running $\left((a_\gamma)_{\gamma=1}^j, i\right) \leftarrow$ Algorithm 1.

[8] Alternatively, in the protocol on Figure 3, after N is computed, U could continue to generate pseudo-random values until it finds one that is coprime with $\varphi(N)$ and then sets it as e. Algorithm U would then also have to reveal the values that did not satisfy this property and prove that they did not, and also to prove that the chosen e and $\varphi(N)$ are coprime. Assuming e to be fixed in advance avoids this complication.

Algorithm 1

```
Require: $\mathsf{PrimeTest}_W$, integers $T, b, e$, pseudo-random function $\mathsf{PRF}$, seed $s$.
Ensure: Pseudo-random numbers $a_\gamma$ and integer $i$.
 1: $ctr, i, j \leftarrow 0$
 2: while $ctr < 2$ and $j < T$ do
 3:     $j \leftarrow j + 1$; $a_j \leftarrow \mathsf{PRF}(s, j)$
 4:     if $\mathsf{PrimeTest}_W\left(b, e, a_j; \mathsf{PRF}(s, j)\right)$ then
 5:         if $ctr = 0$ then
 6:             $i \leftarrow j$
 7:         end if
 8:         $ctr \leftarrow ctr + 1$
 9:     end if
10: end while
11: if $ctr < 2$ then
12:     return $\left((a_\gamma)_{\gamma=1}^{j}, \perp\right)$
13: else
14:     return $\left((a_\gamma)_{\gamma=1}^{j}, i\right)$
15: end if
```

(a) if $i = \perp$ (i.e., Algorithm 1 did not find 2 primes such that $\mathsf{PrimeTest}_W(b, e, a_j; \mathsf{PRF}(s, j)) = 1$ in T iterations; this case is not depicted on Figure 3), U sends $\left(r_{\mathcal{U}}, (a_\gamma)_{\gamma=1}^{j}\right)$ to $\mathcal{CA}$, returns $(0, 0)$ and sets $term_{\mathcal{U}} \leftarrow$ TRUE

(b) if $i \neq \perp$, U computes a proof π that it correctly performed its computation with Π, and a proof π_W that the RSA public key is in $\mathcal{L}_W$ with Π_W. After computing the proofs, U erases all variables but N, e, p, q, i, π, π_W and $(a_\gamma)_{\gamma \neq i,j}$. It sends these latter to $\mathcal{CA}$, except p and q, returns $(pk_{\mathcal{U}} \leftarrow (N, e), sk \leftarrow (p, q, e))$, and sets $term_{\mathcal{U}} \leftarrow$ TRUE

4a. CA, upon receiving $\left(r_{\mathcal{U}}, (a_\gamma)_{\gamma=1}^{j}\right)$ from $\mathcal{U}$, computes $r'_{\mathcal{U}}$, $\rho_{\mathcal{U}}$ and s as U, computes $(C', d') \leftarrow \mathsf{Com}\left(r'_{\mathcal{U}}, \rho_{\mathcal{U}}\right)$, and verifies that $C' = C$ and that $\mathsf{PRF}(s, \gamma) = a_\gamma$ for all $\gamma \in [\![j]\!]$. If all verifications succeed, CA returns 0, otherwise it returns $\perp$. It sets $term_{\mathcal{CA}} \leftarrow$ TRUE

4b. CA, upon receiving $\left(N, e, \pi, \pi_W, i, (a_\gamma)_{\gamma \neq i,j}\right)$ from $\mathcal{U}$, generates a seed s'' with $\mathcal{H}$ from its randomness, and uses it to generate the randomness necessary to compute $\mathsf{PrimeTest}_W$. The resulting random string is denoted r'_W. It verifies that for all $\gamma \in [\![j-1]\!] \setminus \{i\}$, $\mathsf{PrimeTest}_W\left(b, e, a_\gamma; r'_W\right) = 0$, and that π and π_W are valid. If one of the verifications did not succeed, CA returns $\perp$, otherwise it returns $pk_{\mathcal{CA}} \leftarrow (N, e)$. It sets $term_{\mathcal{CA}} \leftarrow$ TRUE.

Discussion. U proves among other things that p and q are really the first two random primes in W such that $\gcd(e, p-1) = \gcd(e, q-1) = 1$, and therefore cannot have chosen primes with additional conditions to these. It is a subtle but crucial aspect of the protocol which ensures that U cannot bias the distribution of the keys; and not doing so is precisely what allowed for the ROCA vulnerabilities [25] in which specific primes where chosen by the user algorithm.

$$
\begin{array}{lcr}
\mathsf{U}\,(pp, e; r_{\mathcal{U}}) & & \mathsf{CA}\,(pp; r_{C\mathcal{A}}) \\
r'_{\mathcal{U}} \leftarrow \mathcal{H}\,(0\|r_{\mathcal{U}});\ \rho_U \leftarrow \mathcal{H}\,(1\|r_{\mathcal{U}}) & & \\
(C, d) \leftarrow \mathsf{Com}\left(r'_{\mathcal{U}}; \rho_U\right) & & \\
s' \leftarrow \mathcal{H}\,(2\|r_{\mathcal{U}});\ r_{\Pi_C} \leftarrow \mathsf{PRF}'\left(s', 0, 1^{\left|r_{\Pi_C}\right|}\right) & & \\
\pi_C \leftarrow \Pi_C.\mathsf{Prove}\left(pp_{\Pi_C}, C, \left(r'_{\mathcal{U}}, d\right); r_{\Pi_C}\right) & & \\
 & \xrightarrow{C, \pi_C} & \Pi_C.\mathsf{Verf}\left(pp_{\Pi_C}, C, \pi_C\right) \overset{?}{=} 1 \\
 & \xleftarrow{r_{C\mathcal{A}}} & \\
s \leftarrow r'_{\mathcal{U}} \oplus \mathcal{H}(r_{C\mathcal{A}}) & & \\
\left((a_\gamma)_{\gamma=1}^{j}, i\right) \leftarrow \text{Alg.1 with random string} & & \\
\quad \mathsf{PRF}'\left(s', \gamma, 1^{|r_W|}\right) \text{ for } \mathsf{PrimeTest}_W & & \\
p \leftarrow a_i,\ q \leftarrow a_j,\ N \leftarrow pq & & \\
 & & \\
\pi \leftarrow \Pi \text{ proof of correct computation with} & & \\
\quad \text{random string } \mathsf{PRF}'\left(s', j+1, 1^{|r_\Pi|}\right) & & \\
\pi_W \leftarrow \Pi_W \text{ proof that } (N, e) \in \mathcal{L}_W \text{ with} & & \\
\quad \text{random string } \mathsf{PRF}'\left(s', j+2, 1^{\left|r_{\Pi_W}\right|}\right) & & \\
 & & \\
\text{Erase all variables but } N, e, i, p, q & & s'' \leftarrow \mathcal{H}(r_{C\mathcal{A}}) \\
\quad (a_\gamma)_{\gamma \neq i,j}, \pi \text{ and } \pi_W & \xrightarrow[i, (a_\gamma)_{\gamma \neq i,j}]{(N,e), \pi, \pi_W} & r'_W \leftarrow \mathsf{PRF}'\,(s'', 0) \\
 & & \forall \gamma \neq i, j,\ \mathsf{PrimeTest}_W\left(b, e, a_\gamma; r'_W\right) \overset{?}{=} 0 \\
 & & \Pi.\mathsf{Verf}\left(pp_\Pi, \left(C, r_{C\mathcal{A}}, N, (a_\gamma)_{\gamma \neq i,j}\right), \pi\right) \overset{?}{=} 1 \\
 & & \Pi_W.\mathsf{Verf}\left(pp_{\Pi_W}, (N, e), \pi_W\right) \overset{?}{=} 1 \\
\text{return } ((N, e), (p, q, e)) & & \text{return } (N, e)
\end{array}
$$

Fig. 3. RSA-Key Generation Protocol with Verifiable Randomness for an Arbitrary Relation $\mathcal{R}_W$.

Correctness & Indistinguishability. Let j be the number of iterations of Algorithm 1. In the full version [6], we show that, if $\mathsf{PrimeTest}_W$ is δ-correct and if PRF' is a secure PRF, $\mathsf{IKG}_{\mathrm{RSA}}$ is approximately $(1 - j(1 - \delta))$-correct in the random-oracle model w.r.t. the class of algorithms that make few oracle queries compared to the min-entropy of the distributions they provide. Moreover, $\mathsf{IKG}_{\mathrm{RSA}}$ is indistinguishable from $\mathsf{KeyGen}_{\mathrm{RSA}}$ in the random oracle if $\mathscr{C}$ is hiding and binding, if Π_C and Π are zero-knowledge and extractable, if Π_W is zero-knowledge and sound, if PRF and PRF' are secure PRFs, if the probability that Algorithm 1 fails although $\mathsf{IKG}_{\mathrm{RSA}}$ does not is small, and if the adversary makes few random-oracle queries compared to the min-entropy of the test distributions. Lastly, in [6], we also show that by tuning the running time of Algorithm 1 depending on the number of primes in the range of PRF that satisfy the conditions on p and q, the probability that Algorithm 1 fails although $\mathsf{IKG}_{\mathrm{RSA}}$ does not is small.

5 Instantiation of the RSA-Key Generation Protocol

In this section, we instantiate the protocol of Section 4.2 for RSA key-generation with verifiable randomness. To do so, we provide efficient instantiations for each of the building blocks.

Recently, several important advancements have been made on the efficiency of the commit-and-prove paradigm on committed values which combine algebraic and non-algebraic statements [14,10,2]. These improvements for *cross-domains* statements allow to prove efficiently for instance that some committed value corresponds to a pre-image of some value of a given hash function such as SHA-256 or that some value is the output of some non-algebraic PRF (i.e. HMAC-SHA-256 or AES) using some committed key. To generate an RSA modulus of 3072 bits (for 128-bit security) using the generic protocol from Section 4.2, the PRF must return 1536-bit integers and the use of non-algebraic PRF with the technique from [14,10,2] would result in prohibitive schemes.

On this account, we present an instantiation based on an algebraic PRF, namely the Dodis–Yampolskiy PRF, and use techniques [10] due to Bünz, Bootle, Boneh, Poelstra, Wuille and Maxwell for range proofs and arithmetic-circuit satisfiability to obtain short proofs of correct computation (i.e., Π in Section 4.2).

In the process, we give the first logarithmic-size (in the bit-length of the group order) argument of knowledge of double discrete logarithms, and argument of equality of a discrete logarithm in a group and a double discrete logarithm in another related group. In contrast, the protocol of Camenisch and Stadler [12] for the first relation, and the protocol of Chase et al. [14] for the second are linear in the security parameter.

Parameters. We consider two related group-family generators $\mathsf{GroupGen}_1$ and $\mathsf{GroupGen}_2$. Given a security parameter λ, to generate an RSA modulus which is the product of two $b(\lambda)$-bit prime numbers, let ℓ be the smallest prime of binary length equal to $b(\lambda)$ such that $2\ell + 1$ is also a prime number (i.e., ℓ is a Sophie Germain prime number, or equivalently $2\ell+1$ is a $b(\lambda)+1$-bit *safe prime*). $\mathsf{GroupGen}_2$ returns, on input λ, the group $\mathbb{G}_2$ of quadratic residues modulo $2\ell+1$ (which is of prime order ℓ). The group-family generator $\mathsf{GroupGen}_1$ returns on input λ some group $\mathbb{G}_1$ of prime order Λ such that ℓ divides $\Lambda - 1$ and $\Lambda > (2\ell + 1)^2$. In practice[9], $\mathbb{G}_1$ can be taken as a prime order subgroup Λ of $\mathbb{Z}_r^*$ for some prime number r such that Λ divides $r - 1$. The restriction to quadratic residues is necessary for assumptions like the q-DDHI assumptions to hold over $\mathbb{G}_2$. However, it introduces a bias by design (not from the user algorithm) in the RSA keys generated: p and q are necessarily quadratic residues modulo $2\ell + 1$. The reason is that the values returned by the DY PRF are not actually integers but $\mathbb{G}_2$ elements. Nonetheless, it is already the case for 1/4 of all RSA moduli since the factors p and q returned by $\mathsf{KeyGen}_{\mathrm{RSA}}$.

[9] To generate RSA moduli which are products of two 1536-bit primes, one possible instantiation for the Dodis–Yampolskiy PRF is to use $\ell = 2^{1535} + 554415$ which is a Sophie Germain prime, $\Lambda = (4\ell + 18)\ell + 1$ and $r = 1572 \cdot \Lambda + 1$.

Commitment Scheme. Scheme $\mathscr{C}$ is the Pedersen commitment scheme [27] in $\mathbb{G}_2$ for the user to commit to her randomness and to the secret p and q.

Pseudo-Random Functions. PRF is the Dodis–Yampolskiy (DY) PRF (see Section 2) in the group $\mathbb{G}_2 = QR_{2\ell+1}$ of quadratic residues modulo $2\ell + 1$. It is used to generate the secret RSA primes p and q. Since $2\ell + 1$ is $b(\lambda) + 1$ bits long, p and q are $b(\lambda)$ bits long with probability close to $1/2$. The reason $2\ell + 1$ is chosen to be one bit larger than p and q is to ensure that *all* primes of $b(\lambda)$ bits can be returned by the PRF so as not to introduce a bias. As for PRF$'$, it can be any efficient pseudo-random function, e.g., HMAC [3].

Argument for R_W. The argument system Π_W depends on the properties that the prime factors of N must satisfy, e.g., they must be congruent to 3 modulo 4 or be safe primes. To prove that $p = q = 3 \mod 4$, one can prove that N is of the form $p^r q^s$ with $p = q = 3 \mod 4$ using the protocol of van de Graaf and Peralta [31], and run in parallel the protocol of Boyar et al. [9] to prove that N is square-free. To prove that p and q are safe primes, there exist proof systems in the literature such as Camenisch and Michel's [11]. Besides, Goldberg et al. [19] recently built a protocol to prove that $\gcd(e, \phi(N)) = 1$.

Argument of Correct Computation. The last component is an extractable zero-knowledge argument system Π in the random-oracle model for the user algorithm to prove that it correctly performed its computation, i.e., an argument system for $\mathcal{R}_\Pi$. Section 5.1 presents a perfectly honest-verifier zero-knowledge interactive protocol for $\mathcal{R}_\Pi$ that is also extractable in the random-oracle model.

5.1 Zero-Knowledge Argument with the Dodis–Yampolskiy PRF

This section gives a zero-knowledge argument Π in the case of the DY PRF in $\mathbb{G}_2 = QR_{2\ell+1}$. Formally, let $2\ell+1$ be a $b(\lambda)+1$-bit (i.e., $b(\lambda)+1 = \lfloor \log(2\ell+1) \rfloor + 1$) safe prime (i.e., ℓ is a Sophie Germain prime) and let Λ be a prime integer such that ℓ divides $\Lambda - 1$ and $\Lambda > (2\ell+1)^2$. Consider $\mathbb{G}_1 = \langle G_1 \rangle$ a group of prime order Λ (in which p and q will be committed) and $\mathbb{G}_2 = \langle G_2 \rangle = QR_{2\ell+1}$ the group of quadratic residues modulo $2\ell + 1$, which is a cyclic group of order ℓ. Recall that the DY PRF is defined as the map $(K, x) \mapsto G_2^{1/(K+x)}$.

Proof Strategy. To prove knowledge of a witness for the membership of $\left(C, r_{C\mathcal{A}}, N, (a_\gamma)_{\gamma \neq i,j}\right)$ to the language relative to $\mathcal{R}_\Pi$, the user algorithm commits to $p = a_i$ and $q = a_j$ in $\mathbb{G}_1$ with the Pedersen commitment scheme and respective randomness r_p and r_q. The commitments are denoted P and Q.

The user algorithm then proves knowledge of a witness for $\mathcal{R}_0 \cap \mathcal{R}_1$, with

$$\begin{aligned}\mathcal{R}_0 := \Big\{ & (C, r_{C\mathcal{A}}, N, P, Q, (a_\gamma)_{\gamma \neq i,j}; r'_{\mathcal{U}}, \rho_u, a_i, a_j, r_p, r_q) : \\ & \mathsf{ComVf}(C, r'_{\mathcal{U}}, \rho_u) = 1,\ s = r'_{\mathcal{U}} + \mathcal{H}(r_{C\mathcal{A}}) \bmod \ell \\ & \forall \gamma \in [\![j]\!], a_\gamma = \mathsf{PRF}(s, \gamma), \mathsf{ComVf}(P, a_i, r_p) = \mathsf{ComVf}(Q, a_j, r_q) = 1 \Big\}\end{aligned}$$

and

$$\mathcal{R}_1 := \Big\{(C, r_{C\mathcal{A}}, N, P, Q, (a_\gamma)_{\gamma \neq i,j}; r'_{\mathcal{U}}, \rho_u, a_i, a_j, r_p, r_q) \colon \mathsf{ComVf}(P, a_i, r_p) = 1$$
$$\mathsf{ComVf}(Q, a_j, r_q) = 1, 2^{b(\lambda)-1} \leq a_i, a_j \leq 2^{b(\lambda)} - 1, N = a_i a_j \text{ in } \mathbb{N}\Big\}.$$

To prove knowledge of a witness for relation $\mathcal{R}$, it then suffices to prove in parallel knowledge of a witness for $\mathcal{R}_0$ and of a witness for $\mathcal{R}_1$ on the same public inputs. Note that the binding property of the Pedersen commitment scheme in $\mathbb{G}_1$ (relying on the DLOG assumption) guarantees that the a_i and a_j values used in both proofs are the same (up to a relabeling).

Relation $\mathcal{R}_0$. We start by giving two preliminary protocols:

– a logarithmic-size zero-knowledge argument of knowledge of a double-discrete logarithm (Section 5.2) using Bulletproof techniques [10]. The resulting proofs are of size logarithmic in the bit-length of the group order. In comparison, the protocol of Camenisch and Stadler [12] has proofs of size linear in the security parameter
– a logarithmic-size argument of equality of a discrete logarithm in a group and a double discrete logarithm in another related group (Section 5.2). In contrast, the protocol of Chase et al. [14, Section 4.3] for this relation uses the techniques of Camenisch and Stadler and therefore has proofs of size linear in the security parameter.

We then combine the latter proof with the proof in Section 5.3 to obtain a proof for relation $\mathcal{R}_0$.

Relation $\mathcal{R}_1$. The aggregated logarithmic range proof of Bünz et al. [10, Section4.2] is sufficient to prove that the values committed in P and Q modulo Λ are in $\left[\!\left[2^{b-1}, 2^b - 1\right]\!\right]$ (which is equivalent to proving that the values committed in $PG_1^{-2^{b-1}}$ and $QG_1^{-2^{b-1}}$ are in $\left\{0, \ldots, 2^{b-1} - 1\right\}$). With the hypotheses on the parameters Λ and ℓ, the verifier is convinced that the equation $N = a_i a_j$ holds in $\mathbb{N}$. Indeed, the equation $N = a_i a_j \mod \Lambda$ implies that there exists $m \in \mathbb{Z}$ such that $N = a_i a_j + m\Lambda$. Integer m cannot be strictly positive as otherwise N would be strictly greater than Λ. Besides, m cannot be strictly negative since $\Lambda > (2\ell + 1)^2 > a_i a_j$; it is therefore nil and the equation $N = a_i a_j$ holds in $\mathbb{N}$.

5.2 Logarithmic-Size Argument of Double Discrete Logarithm

This section gives a zero-knowledge argument with logarithmic communication size for proving knowledge of a double discrete logarithm. It uses as a sub-argument the logarithmic-size inner-product argument for arithmetic-circuit satisfiability of Bünz et al. [10, § 5.2], which is complete, perfectly honest-verifier zero-knowledge, and satisfies witness-extended emulation.

Following the ideas of Bootle et al. [8], Bünz et al. convert any arithmetic circuit with n multiplications gates into a Hadamard product $\boldsymbol{a}_L \circ \boldsymbol{a}_R = \boldsymbol{a}_O$ and $Q \leq 2n$ linear constraints of the form

$$\langle \boldsymbol{w}_{L,q}, \boldsymbol{a}_L \rangle + \langle \boldsymbol{w}_{R,q}, \boldsymbol{a}_R \rangle + \langle \boldsymbol{w}_{O,q}, \boldsymbol{a}_O \rangle = c_q$$

for $q \in [\![Q]\!]$, with $\boldsymbol{w}_{L,q}, \boldsymbol{w}_{R,q}, \boldsymbol{w}_{O,q} \in \mathbb{Z}_p^n$ and $c_q \in \mathbb{Z}_p$. The vectors $\boldsymbol{a}_L$, $\boldsymbol{a}_R$ respectively denote the vectors of left and right inputs to the multiplications gates, and $\boldsymbol{a}_O$ the vector of outputs. The linear constraints ensure the consistency between the outputs and the inputs of two consecutive depth levels of the circuit. Bootle et al. [8, App. A] give a general method to find such linear constraints, though it may not always result in the most compact ones for a specific circuit.

Bünz et al. actually give an argument for a more general relation which includes Pedersen commitments of which the openings are included in the linear consistency constraints. Concretely, given a group $\mathbb{G}$ of prime order p and positive integers n, m and Q, Bünz et al. give a zero-knowledge argument for the relation

$$\begin{aligned}\{(g,\ \ & h \in \mathbb{G}, \mathbf{g}, \mathbf{h} \in \mathbb{G}^n, \mathbf{V} \in \mathbb{G}^m, \mathbf{W}_L, \mathbf{W}_R, \mathbf{W}_O \in \mathbb{Z}_p^{Q\times n}, \mathbf{W}_V \in \mathbb{Z}_p^{Q\times m},\\ & \mathbf{c} \in \mathbb{Z}_p^Q; \mathbf{a}_L, \mathbf{a}_R, \mathbf{a}_O \in \mathbb{Z}_p^n, \mathbf{v}, \gamma \in \mathbb{Z}_p^m\big) : V_j = g^{v_j} h^{\gamma_j} \forall j \in [\![m]\!]\\ & \wedge \mathbf{a}_L \circ \mathbf{a}_R = \mathbf{a}_O \wedge \mathbf{W}_L \mathbf{a}_L + \mathbf{W}_R \mathbf{a}_R + \mathbf{W}_O \mathbf{a}_O = \mathbf{W}_V \mathbf{v} + \mathbf{c}\}\,.\end{aligned}$$

Its soundness relies on the discrete-logarithm assumption over the generator of $\mathbb{G}$ and the prover sends $2\lceil \log_2 n \rceil + 8$ group elements and 5 $\mathbb{Z}_p$ elements.

The main difficulty in the case of a proof of a double discrete logarithm relation is to re-write the problem in a way that is suitable to apply the proof for arithmetic circuits. The goal is to give a zero-knowledge argument for:

$$\mathcal{R}_{\mathrm{2DLOG}} := \left\{ (G_1, H_1, G_2, Y; x \in \mathbb{Z}_\ell, r \in \mathbb{Z}_\Lambda) \colon Y = G_1^{G_2^x} H_1^r \right\}.$$

First, let $n(\lambda) + 1 := b(\lambda)$ be the bit-length of ℓ. Given the bit representation $(x_i)_{i=0}^n$ of x, $G_2^x = G_2^{\sum_{i=0}^n x_i 2^i} = \prod_i \left(G_2^{2^i}\right)^{x_i}$. An important observation is that for $x_i \in \{0,1\}$, $\left(G_2^{2^i}\right)^{x_i} = x_i G_2^{2^i} + (1 - x_i) = x_i \left(G_2^{2^i} - 1\right) + 1$. The addition here is over $\mathbb{Z}_\Lambda$, although the notation is purely formal since $x_i \in \{0,1\}$. It thus follows that an argument for $\mathcal{R}_{\mathrm{2DLOG}}$ is equivalent to an argument for:

$$\left\{ (G_1, H_1, G_2, Y; (x_i)_{i=0}^n \in \{0,1\}^n, r \in \mathbb{Z}_\Lambda) \colon Y = G_1^{\prod_i \left(x_i \left(G_2^{2^i} - 1\right) + 1\right)} H_1^r \right\},$$

which is also equivalent to an argument for:

$$\left\{ (G_1, H_1, G_2, Y; (a_i)_{i=0}^n, r \in \mathbb{Z}_\Lambda) \colon Y = G_1^{\prod_i a_i} H_1^r \wedge a_i \in \left\{1, G_2^{2^i}\right\} \right\}.$$

To this end, consider the following array

$$\begin{array}{ccccc} a_0 & a_1 & a_2 & \cdots & a_n \\ 1 & a_0 & a_0 a_1 & \cdots & a_0 a_1 \cdots a_{n-1} \\ a_0 & a_0 a_1 & a_0 a_1 a_2 & \cdots & a_0 \cdots a_n. \end{array}$$

Notice that its third row is the product of the first two. In other words, if $\mathbf{a} \leftarrow (a_0, a_1, \ldots, a_n) \in \mathbb{Z}_\Lambda^{n+1}$ and $\mathbf{b} \leftarrow (b_0 = a_0, b_1 = a_0 a_1, \ldots, b_{n-1} = a_0 a_1 \cdots a_{n-1}) \in \mathbb{Z}_\Lambda^n$, then $\mathbf{a} \circ (1 \quad \mathbf{b}) = (\mathbf{b} \quad y)$ for $y := G_2^x$.

$$\underbrace{\begin{bmatrix} \mathbf{0}_{(n+2)\times 2(n+1)} \\ \mathbf{I}_{n+1} \quad -\mathbf{I}_{n+1} \\ \mathbf{I}_{n+1} \; \mathbf{0}_{(n+1)\times(n+1)} \\ \mathbf{0}_{(n+1)\times 2(n+1)} \end{bmatrix}}_{\mathbf{W}_L} \underbrace{\begin{bmatrix} \mathbf{U}_{n+1} \; \mathbf{0}_{(n+1)\times(n+1)} \\ \mathbf{E}_{1\times 2(n+1)}(1,1) \\ \mathbf{0}_{(n+1)\times 2(n+1)} \\ \mathbf{0}_{(n+1)\times(n+1)} \; -\mathbf{I}_{n+1} \\ \mathbf{0}_{(n+1)\times 2(n+1)} \end{bmatrix}}_{\mathbf{W}_R} \underbrace{\begin{bmatrix} -\mathbf{I}_n \; \mathbf{0}_{n\times(n+2)} \\ \mathbf{E}_{2\times 2(n+1)}(1,n+1) \\ \mathbf{0}_{2(n+1)\times 2(n+1)} \\ \mathbf{0}_{(n+1)\times(n+1)} \; \mathbf{I}_{n+1} \end{bmatrix}}_{\mathbf{W}_O} \underbrace{\begin{bmatrix} \mathbf{0}_{n\times 2} \\ 0 \; 1 \\ 1 \; 0 \\ \mathbf{0}_{3(n+1)\times 2} \end{bmatrix}}_{\mathbf{W}_V}$$

Fig. 4. Matrices $\mathbf{W}_L$, $\mathbf{W}_R$, $\mathbf{W}_O$ and $\mathbf{W}_V$ for the proof of double discrete logarithm. $\mathbf{U}_{n+1}$ denotes the square $n+1$-matrix with 1 on the upper diagonal. $\mathbf{E}_{n\times m}(i,j)$ denotes the $n \times m$ matrix with 1 at position (i,j) and 0 elsewhere.

Moreover, for $\mathbf{a}_L := \left[\mathbf{a}\ \mathbf{a} - \mathbf{1}^{n+1}\right]^T$, $\mathbf{a}_R := \left[1\ \mathbf{b}\ \mathbf{a} - \mathbf{G}_2^{\mathbf{2}^{n+1}}\right]^T$ and $\mathbf{a}_O := \left[\mathbf{b}\ y\ \mathbf{0}^{n+1}\right]^T \in \mathbb{Z}_\Lambda^{2(n+1)}$ where $\mathbf{G}_2^{\mathbf{2}^{n+1}}$ denotes the vector $\left(G_2, G_2^2, G_2^{2^2}, \ldots, G_2^{2^n}\right)$, one has $\mathbf{a}_L \circ \mathbf{a}_R = \mathbf{a}_O$. If one can prove knowledge of scalars $y, r \in \mathbb{Z}_\Lambda$ and of vectors $\mathbf{a}_L$, $\mathbf{a}_R$ and $\mathbf{a}_O$ such that $Y = G_1^y H_1^r$ and $\mathbf{a}_L \circ \mathbf{a}_R = \mathbf{a}_O$, and such that the vectors are of the form above, then one can prove knowledge of $(a_i)_{i=0}^n \in \prod_i \left\{1, G_2^{2^i}\right\}$ and $(b_i)_{i=0}^{n-1}$ such that $y = a_n b_{n-1} = a_n a_{n-1} b_{n-2} = \cdots = a_n a_{n-1} \cdots a_1 b_0 = a_n \cdots a_0$ and $Y = G_1^y H_1^r$. That is to say, one can prove knowledge of a double discrete logarithm.

To prove such a relation, one can use the argument of Bünz et al. [10] for arithmetic circuits with the right linear constraints to ensure that the vectors are of the appropriate form. To express these constraints, consider matrices $\mathbf{W}_L$, $\mathbf{W}_R$, $\mathbf{W}_O$ and $\mathbf{W}_V$ from Figure 4 and vectors $\mathbf{v}^T := \left[1\ y\right]$, $\mathbf{c}^T := \left[\mathbf{0}_{1\times(n+2)}\ \mathbf{1}^{n+1}\ \mathbf{G}_2^{\mathbf{2}^{n+1}}\ \mathbf{0}_{1\times(n+1)}\right]$.

Three vectors $\mathbf{a}_L$, $\mathbf{a}_R$ and $\mathbf{a}_O \in \mathbb{Z}_\Lambda^{2(n+1)}$ satisfy the equation $\mathbf{W}_L\mathbf{a}_L + \mathbf{W}_R\mathbf{a}_R + \mathbf{W}_O\mathbf{a}_O = \mathbf{W}_V\mathbf{v} + \mathbf{c}$ if and only if there exists $\mathbf{a} \in \mathbb{Z}_\Lambda^{n+1}$ and $\mathbf{b} \in \mathbb{Z}_\Lambda^n$ such that $\mathbf{a}_L^T := \left[\mathbf{a}\ \mathbf{a} - \mathbf{1}^{n+1}\right]$, $\mathbf{a}_R^T := \left[1\ \mathbf{b}\ \mathbf{a} - \mathbf{G}_2^{\mathbf{2}^{n+1}}\right]$ and $\mathbf{a}_O^T := \left[\mathbf{b}\ y\ \mathbf{0}^{n+1}\right] \in \mathbb{Z}_\Lambda^{2(n+1)}$ (see [6]).

The argument of Bünz et al. is therefore sufficient to prove in zero-knowledge knowledge of a double discrete logarithm. The soundness of the proof relies on the discrete-logarithm assumption over $\mathbb{G}_1$.

Regarding the proof size, the prover sends $(2\lceil\log_2 2(n+1)\rceil + 8)$ $\mathbb{G}_1$ elements and 5 $\mathbb{Z}_\Lambda$ elements. Notice that the argument of Bünz et al. requires $4(n+1)$ elements of $\mathbb{G}_1^*$ in addition to G_1 and H_1. To guarantee its soundness, no discrete logarithm relation between these elements, G_1 and H_1 must be known to the prover. They can then be choosen uniformly at random during set-up.

Logarithmic-Size Argument of Discrete Logarithm and Double Discrete Logarithm Equality. Building on the argument of double discrete logarithm of Section 5.2, we present in [6] a zero-knowledge argument for the relation

$$\mathcal{R}_{\mathrm{DLOG}-2} := \left\{(G_1, H_1, G_2, H_2, Y, X; x \in \mathbb{Z}_\ell, r_1, r_2 \in \mathbb{Z}_\Lambda) \colon Y = G_1^{G_2^x} H_1^{r_1}, X = G_2^x H_2^{r_2}\right\}.$$

In total, the prover sends $2\lceil\log_2 4(n+1)\rceil + 8$ $\mathbb{G}_1$ elements and 5 $\mathbb{Z}_\Lambda$ elements.

5.3 An Intermediate Protocol in $\mathbb{G}_2$

This section gives an perfect honest verifier zero-knowledge protocol for relation

$$\begin{aligned}\mathcal{R}_0' \coloneqq \Big\{ \big(G_2, H_2, X_p, X_q, U, K_u, (K_\gamma)_{\gamma \neq i,j}, (a_\gamma)_\gamma ; x_p, x_q, r_p, r_q, u, \rho_u \big) :\\ \forall \pi \in \{p, q\}, X_\pi = G_2^{x_\pi} H_2^{r_\pi}, U = G_2^u H_2^{\rho_u},\\ \forall \gamma, a_\gamma = G_2^{x_\gamma}, x_\pi(u + K_\pi) = x_\gamma(u + K_\gamma) = 1 \mod \ell \Big\} .\end{aligned}$$

Note that for $\pi \in \{p, q\}$, $\left(U G_2^{K_\pi}\right)^{x_\pi} H_2^{-x_\pi \rho_u} = G_2$, and that $\forall \gamma$, $a_\gamma^u = G_2 a_\gamma^{-K_\gamma}$, i.e., the discrete logarithms of $G_2 a_\gamma^{-K_\gamma}$ in base a_γ for all γ are the same.

The protocol is given on Figure 5). As the proof system is public-coin, it can be made non-interactive in the random-oracle model via the Fiat–Shamir heuristic by computing c as $\mathcal{H}(G_2, H_2, (X_\pi), U, (K_\pi), (K_\gamma)_\gamma, (a_\gamma)_\gamma, (Y_\pi), V, (H_\pi), (A_\gamma))$ for a random oracle $\mathcal{H}$ with $\mathbb{Z}_\ell$ as range. The proof then consists of $(c, (z_\pi, t_\pi)_{\pi \in \{p,q\}}, w, \tau_u, (\tau_\pi)_\pi)$, i.e., 9 $\mathbb{Z}_\ell$ elements.

The protocol is complete, perfectly honest-verifier zero-knowledge, and satisfies witness-extended emulation under the discrete logarithm assumption over $\mathbb{G}_2$. The protocol completeness and its zero-knowledge property are straightforward. See the full version [6] for the proof of the witness-extended–emulation.

$$\boxed{\begin{array}{lcr}
\mathscr{P}(\cdots ; (x_\pi)_{\pi \in \{p,q\}}, (r_\pi), u, \rho_u) & & \mathscr{V}\big(G_2, H_2, (X_\pi), U, (K_\pi), (K_\gamma)_\gamma, (a_\gamma)_\gamma\big) \\
y_\pi, s_\pi, v, \sigma_u, \sigma \leftarrow_\$ \mathbb{Z}_\ell & & \\
Y_\pi \leftarrow G_2^{y_\pi} H_2^{s_\pi} & & \\
V \leftarrow G_2^{v} H_2^{\sigma_u} & \xrightarrow{(Y_\pi)_\pi, V, (H_\pi)_\pi, (A_\gamma)_\gamma} & \\
H_\pi \leftarrow \left(U G_2^{K_u}\right)^{y_\pi} H_2^{\sigma_\pi} & & \\
A_\gamma \leftarrow \left(a_\gamma\right)^{v} & & \\
 & & c \leftarrow_\$ \mathbb{Z}_\ell \\
 & \xleftarrow{c} & \\
z_\pi \leftarrow y_\pi - c x_\pi, t_\pi \leftarrow s_\pi - c r_\pi & & G_2^{z_\pi} H_2^{t_\pi} X_\pi^c \overset{?}{=} Y_\pi \\
w \leftarrow v - cu, \tau_u \leftarrow \sigma_u - c\rho_u & \xrightarrow[w, \tau_u, (\tau_\pi)_\pi]{(z_\pi, t_\pi)_\pi} & G_2^{w} H_2^{\tau_u} U^c \overset{?}{=} V \\
\tau_\pi \leftarrow \sigma_\pi + c x_\pi \rho_u & & \left(U G_2^{K_u}\right)^{z_\pi} H_2^{\tau_\pi} G_2^c \overset{?}{=} H_\pi \\
 & & a_\gamma^w \left(G_2 a_\gamma^{-K_\gamma}\right)^c \overset{?}{=} A_\gamma
\end{array}}$$

Fig. 5. Honest-Verifier Zero-Knowledge Protocol for Relation $\mathcal{R}_1$.

5.4 Protocol for $\mathcal{R}_0$

To prove knowledge of a witness for $\mathcal{R}_0$, the prover starts setting by setting $K_p \coloneqq \mathcal{H}(r_{C\mathcal{A}}) + i$, $K_q \coloneqq \mathcal{H}(r_{C\mathcal{A}}) + j$, and $u \coloneqq r'_{\mathcal{U}}$. It then

– computes two commitments $X_p = G_2^{x_p} H_2^{r_p}$ and $X_q = G_2^{x_q} H_2^{r_q}$, for $x_p = (u + K_p)^{-1} \mod \ell$ and $x_q = (u + K_q)^{-1} \mod \ell$

– computes a proof $\pi_{\mathrm{DLOG-2},p}$ that the double discrete-logarithm of P is the discrete logarithm of X_p, and similarly a proof $\pi_{\mathrm{DLOG-2},q}$ for Q and X_q
– computes a proof π' for relation $\mathcal{R}'_0$ with X_p and X_q.

The final proof π_0 for $\mathcal{R}_0$ consists of $\left(X_p, X_q, \pi_{\mathrm{DLOG-2},p}, \pi_{\mathrm{DLOG-2},q}, \pi'_0\right)$.

Security. It is important to note that the security of the generated key is weakened compared to an RSA-key of the same size since the CA can recover seed s (and thus the prime factors) by solving a discrete logarithm problem in $\mathbb{G}_2$. For 3072-bit RSA moduli, this protocol therefore only provides 96 bits of security (with respect to the CA) instead of the expected 128-bit security level. To avoid this issue, one can increase the bit size of the prime numbers to 3072 bits (but at the cost of generating RSA moduli of twice this size). Another possibility is to use other groups for $\mathbb{G}_2$ with (alleged) harder discrete logarithm problem, e.g., the group of points of an elliptic curve over $\mathbb{F}_p$ or an algebraic torus defined over $\mathbb{F}_{p^2}$ (with compact representation of group elements in $\mathbb{F}_p$) for a 1536-bit prime p. This may however introduce a new bias for the generated primes and require to adapt the zero-knowledge proofs.

Efficiency. The asymptotic complexity of the communication size depends on the number of trials to obtain two primes in W since the prover has to send $(a_\gamma)_{\gamma \neq i,j}$. However, even though the communication is asymptotically linear in the number of trials, the overhead incurred by the proof of correct computation should in practice be small.

	$\mathbb{Z}_\ell$	$\mathbb{Z}_N$	$\mathbb{Z}_\Lambda$	$\mathbb{G}_1$	$\mathbb{G}_2$	Total (kB)
$\mathcal{R}_0$	9	0	10	$4\lceil\log_2 4b\rceil + 16$	2	346
$\mathcal{R}_1$	0	0	5	$2\lceil\log_2 2(b-1)\rceil + 4$	0	142

Fig. 6. Size of the Arguments (for a 96-bit security level and 3072-bit RSA moduli).

Total Proof Size. As discussed in Section 5.2, proofs $\pi_{\mathrm{DLOG-2},p}$ and $\pi_{\mathrm{DLOG-2},q}$ both consists of $2\lceil\log_2 4(n+1)\rceil + 8$ $\mathbb{G}_1$ elements and $5\mathbb{Z}_\Lambda$ elements.

Proof π' consists of 9 $\mathbb{Z}_\ell$ elements (see Section 5.3). Proof π_0 for $\mathcal{R}_0$ therefore consists of 2 $\mathbb{G}_2$ elements, $4\lceil\log_2 4(n+1)\rceil + 16$ $\mathbb{G}_1$ elements, 10 $\mathbb{Z}_\Lambda$ elements and 9 $\mathbb{Z}_\ell$ elements. As for the proof for $\mathcal{R}_1$, the aggregated proof that 2 values committed in $\mathbb{G}_1$ are in $\left[\!\left[0, 2^{b-1} - 1\right]\!\right]$ consists of $2\lceil\log_2 2(b-1)\rceil + 4$ $\mathbb{G}_1$ elements (recall that $n + 1 = b$) and 5 $\mathbb{Z}_\Lambda$ elements.

Running Time. An important question about the protocol is the number of necessary PRF trials to obtain two primes that satisfy the conditions required for the factors of N (captured by $W \subseteq \mathbb{P}$). We estimate the number j of necessary trials in the case $W = \mathbb{P} \cap \left[\!\left[2^{b-1}, 2^b - 1\right]\!\right]$, i.e., when $\mathcal{U}$ simply has to prove that p and q are prime of $b(\lambda)$ bits. The full version [6] shows (using a number-theoretic heuristic) that the number of trials exceeds $17b(\lambda) = O(\log \lambda)$ (so the DY PRF remains secure), and that the probability that it is larger than that decreases exponentially fast.

Overall Communication Size. In the last flow of the protocol, the prover then sends an integer N, two commitments in $\mathbb{G}_1$, $17b(\lambda) - 2$ integers in $[\![0, 2\ell]\!]$ with high probability, i.e., the $(a_\gamma)_{\gamma \neq i,j}$ values which are integers returned by the PRF and not in W, and the proof of correct computation of which the size is summarized in Table 6.

Acknowledgements. The authors thank Jonathan Bootle for fruitful discussions. This work was supported by the French ANR ALAMBIC Project (ANR-16-CE39-0006), the CHIST-ERA USEIT project and the EU H2020 Research and Innovation Program under Grant Agreement No. 786725 (OLYMPUS).

References

1. Auerbach, B., Poettering, B.: Hashing solutions instead of generating problems: On the interactive certification of RSA moduli. In: Abdalla, M., Dahab, R. (eds.) PKC 2018, LNCS 10770, pp. 403–430. (2018)
2. Backes, M., Hanzlik, L., Herzberg, A., Kate, A., Pryvalov, I.: Efficient non-interactive zero-knowledge proofs in cross-domains without trusted setup. In: Lin, D., Sako, K. (eds.) PKC 2019, LNCS 11442, pp. 286–313. (2019)
3. Bellare, M., Canetti, R., Krawczyk, H.: Keying hash functions for message authentication. In: Koblitz, N. (ed.) CRYPTO'96. LNCS 1109, pp. 1–15. (1996)
4. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 139–155. (2000)
5. Benhamouda, F., Ferradi, H., Géraud, R., Naccache, D.: Non-interactive provably secure attestations for arbitrary RSA prime generation algorithms. In: Foley, S.N., Gollmann, D., Snekkenes, E. (eds.) ESORICS 2017, LNCS 10492, pp. 206–223. (2017)
6. Blazy, O., Towa, P., Vergnaud, D.: Public-Key Generation with Verifiable Randomness. Cryptology ePrint Archive, Report 2020/294, 2020
7. Boneh, D., Boyen, X.: Short signatures without random oracles. In: Cachin, C., Camenisch, J. (eds.) EUROCRYPT 2004. LNCS 3027, pp. 56–73. (2004)
8. Bootle, J., Cerulli, A., Chaidos, P., Groth, J., Petit, C.: Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, LNCS 9666, pp. 327–357. (2016)
9. Boyar, J., Friedl, K., Lund, C.: Practical zero-knowledge proofs: Giving hints and using deficiencies. In: Quisquater, J.J., Vandewalle, J. (eds.) EUROCRYPT'89. LNCS 434, pp. 155–172. (1990)
10. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy. pp. 315–334. IEEE Computer Society Press (2018)
11. Camenisch, J., Michels, M.: Proving in zero-knowledge that a number is the product of two safe primes. In: Stern, J. (ed.) EUROCRYPT'99. LNCS 1592, pp. 107–122. (1999)
12. Camenisch, J., Stadler, M.: Efficient group signature schemes for large groups (extended abstract). In: Kaliski Jr., B.S. (ed.) CRYPTO'97. LNCS 1294, pp. 410–424. (1997)

13. Canard, S., Gouget, A.: Divisible e-cash systems can be truly anonymous. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS 4515, pp. 482–497. (2007)
14. Chase, M., Ganesh, C., Mohassel, P.: Efficient zero-knowledge proof of algebraic and non-algebraic statements with applications to privacy preserving credentials. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, LNCS 9816, pp. 499–530. (2016)
15. Corrigan-Gibbs, H., Mu, W., Boneh, D., Ford, B.: Ensuring high-quality randomness in cryptographic key generation. In: Sadeghi, A.R., Gligor, V.D., Yung, M. (eds.) ACM CCS 2013. pp. 685–696. ACM Press (2013)
16. Desmedt, Y.: Simmons' protocol is not free of subliminal channels. In: Ninth IEEE Computer Security Foundations Workshop, March 10 - 12, 1996, Dromquinna Manor, Kenmare, County Kerry, Ireland. pp. 170–175 (1996)
17. Dodis, Y., Yampolskiy, A.: A verifiable random function with short proofs and keys. In: Vaudenay, S. (ed.) PKC 2005. LNCS 3386, pp. 416–431. (2005)
18. Gennaro, R., Micciancio, D., Rabin, T.: An efficient non-interactive statistical zero-knowledge proof system for quasi-safe prime products. In: Gong, L., Reiter, M.K. (eds.) ACM CCS 98. pp. 67–72. ACM Press (1998)
19. Goldberg, S., Reyzin, L., Sagga, O., Baldimtsi, F.: Efficient noninteractive certification of RSA moduli and beyond. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019, LNCS 11923, pp. 700–727. (2019)
20. Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions (extended abstract). In: 25th FOCS. pp. 464–479. IEEE Computer Society (1984)
21. Heninger, N., Durumeric, Z., Wustrow, E., Halderman, J.A.: Mining your ps and qs: Detection of widespread weak keys in network devices. In: Kohno, T. (ed.) USENIX Security 2012. pp. 205–220. USENIX Association (2012)
22. Juels, A., Guajardo, J.: RSA key generation with verifiable randomness. In: Naccache, D., Paillier, P. (eds.) PKC 2002. LNCS 2274, pp. 357–374. (2002)
23. Lenstra, A.K., Hughes, J.P., Augier, M., Bos, J.W., Kleinjung, T., Wachter, C.: Public keys. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 626–642. (2012)
24. Mironov, I., Stephens-Davidowitz, N.: Cryptographic reverse firewalls. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, LNCS 9057, pp. 657–686. (2015)
25. Nemec, M., Sýs, M., Svenda, P., Klinec, D., Matyas, V.: The return of coppersmith's attack: Practical factorization of widely used RSA moduli. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 1631–1648. ACM Press (Oct / Nov 2017)
26. NIST: National Institute of Standards and Technology – Digital signature standard (dss). `https://csrc.nist.gov/publications/detail/fips/186/4/final` (2013)
27. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) CRYPTO'91. LNCS 576, pp. 129–140. (1992)
28. Rivest, R.L., Shamir, A., Adleman, L.M.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the Association for Computing Machinery 21(2), 120–126 (1978)
29. Simmons, G.J.: The prisoners' problem and the subliminal channel. In: Chaum, D. (ed.) CRYPTO'83. pp. 51–67. Plenum Press, New York, USA (1983)
30. Stadler, M.: Publicly verifiable secret sharing. In: Maurer, U.M. (ed.) EUROCRYPT'96. LNCS 1070, pp. 190–199. (1996)
31. van de Graaf, J., Peralta, R.: A simple and secure way to show the validity of your public key. In: Pomerance, C. (ed.) CRYPTO'87. LNCS 293, pp. 128–134. (1988)
32. Young, A., Yung, M.: Kleptography: Using cryptography against cryptography. In: Fumy, W. (ed.) EUROCRYPT'97. LNCS 1233, pp. 62–74. (1997)