SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

July 2024, Vol. 67, Iss. 7, 172102:1–172102:12
https://doi.org/10.1007/s11432-023-3883-4

Blockchain-based immunization against kleptographic attacks

Changsong JIANG[1,2], Chunxiang XU[1,2*], Jie CHEN[1,2] & Kefei CHEN[3]

[1]*School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu* 611731*, China;*
[2]*Yangtze Delta Region Institute (Huzhou), University of Electronic Science and Technology of China, Huzhou* 313001*, China;*
[3]*Department of Mathematics, Hangzhou Normal University, Hangzhou* 310027*, China*

Received 29 May 2023/Revised 25 September 2023/Accepted 20 October 2023/Published online 6 June 2024

Abstract Adversarial implementations of cryptographic primitives called kleptographic attacks cause the leakage of secret information. Subliminal channel attacks are one of the kleptographic attacks. In such attacks, backdoors are embedded in implementations of randomized algorithms to elaborately control randomness generation, such that the secrets will be leaked from biased outputs. To thwart subliminal channel attacks, double-splitting is a feasible solution, which splits the randomness generator of a randomized algorithm into two independent generators. In this paper, we instantiate double-splitting to propose a secure randomness generation algorithm dubbed SRG using two physically independent generators: ordinary and public randomness generators. Based on public blockchains, we construct the public randomness generator, which can be verified publicly. Hashes of a sufficient number of consecutive blocks that are newly confirmed on a blockchain are used to produce public randomness. In SRG, outputs from the two generators are taken as inputs of an immunization function. SRG accomplishes immunization against subliminal channel attacks. Additionally, we discuss the application strategies of SRG for symmetric and public-key encryption.

Keywords kleptographic attacks, subliminal channel, blockchain, immunization, randomized algorithm

1 Introduction

Recent years have witnessed a spectacular development of modern cryptography. Notably, cryptographic applications are based on the assumption that the implementations of cryptographic primitives fully conform to their specifications which are proved secure by formal security analysis. However, such an assumption is not always satisfied in the real world. The implementation may deviate from the specification. A common example is a bug in which unintended errors could produce a faulty implementation. Another extreme example is adversarial implementation with backdoors. The subverted implementation is destructive, as it may leak secret information (e.g., secret keys) undetectably.

Such subversion is defined as kleptographic attacks [1, 2]. Recently, the striking Snowden revelations of massive surveillance [3] further reveal the real threat of kleptographic attacks. Moreover, subliminal channel attacks are one of the kleptographic attacks. These attacks stealthily embed a subliminal channel in the output of randomized algorithms [4, 5]. The infected randomized algorithm chooses a desired randomness by rejection-sampling, and then generates a biased output such that secret information can be undetectably leaked to adversaries through the output. Furthermore, an adversary can launch subliminal channel attacks to any target cryptographic scheme containing randomized algorithms, while maintaining the implementation indistinguishable from the fair one (i.e., the implementation that fully conforms to the corresponding specification), even in intensive (black-box) testing [6, 7].

To render cryptographic primitives secure against subliminal channel attacks, a security model dubbed the watchdog model is introduced [8]. The watchdog model relies on the fact that an algorithm can usually be split into functional components executed independently following a trusted amalgamation. A

* Corresponding author (email: chxxu@uestc.edu.cn)

trusted checker called a watchdog will conduct a black-box test to decide whether the implementation of each component is consistent with its specification. According to their supervisory power, watchdogs can be divided into three categories (i.e., offline, online, and omniscient watchdogs). Offline watchdogs can have oracle access to the (possibly subverted) implementation and perform a one-time test before using the implementation. Compared with offline watchdogs, online and omniscient watchdogs are allowed to additionally monitor public interaction between users when the implementation is being used. Omniscient watchdogs are even privy to private information such as users' secret keys. In practical applications, the instantiation of online (or omniscient) watchdogs is costly as they are required to actively monitor all public communications. Therefore, offline watchdogs will be considered in this paper.

According to the watchdog model, a splitting mechanism was proposed [8]. For a randomized algorithm decomposed into a randomness generator (RG) and a deterministic algorithm, a hash function is added as an immunization function tailing RG and hashes output of RG, thereby immunizing the randomized algorithm. However, a sophisticated adversary would surreptitiously insert the immunization function and the deterministic algorithm into RG to generate randomness that satisfies the rejection-sampling condition [9]. Moreover, the adversary inserts a backdoor into RG, so that the corrupted implementation of RG can produce a random coin that leads to the subliminal leakage of confidential data by rejection-sampling.

A feasible solution to this issue is to use double-splitting [9], in which RG is split into two randomness generators, and their outputs are jointly passed to the immunization function. As the output of the immunization function is not determined by either generator, this strategy prevents rejection-sampling, and accordingly achieves immunization against subliminal channel attacks in the watchdog model. Notably, the watchdog model intrinsically relies on the independent execution of an algorithm's functional components. Therefore, two randomness generators of double-splitting as functional components need to be independently executed. This requirement can be satisfied via two physically independent random sources. The design of such random sources for instantiating double-splitting is crucial because double-splitting can be used to immunize cryptographic primitives, such as encryption schemes.

Herein, to produce random sources for instantiating double-splitting, we adopt an original randomness generator as one random source and construct a public randomness generator (PubRG) as the other. We refer to this instantiation as a secure randomness generation algorithm (SRG). In SRG, PubRG is based on public blockchains. Hashes of a sufficient number of consecutive blocks, which are newly confirmed on a blockchain (such as Bitcoin [10] and Ethereum [11]), are used as randomness. The immunization function creates a public randomness using this randomness. According to blockchains' chain quality property [12, 13], the adversary's full control of φ-successive blocks is not likely to occur. Therefore, the probability of undermining public randomness generation is negligible. Our public randomness generator PubRG circumvents subliminal channel attacks, and enables anyone to verify the correctness of public randomness generation based on the chain consistency property of blockchains [14, 15]. Armknecht et al. [16, 17] used blockchains to construct public randomness generators. Their generators adopt a single block, which is the latest confirmed on a blockchain, to produce public randomness. Due to chain quality, the use of φ-successive blocks guarantees that no adversary can control the output of PubRG, unlike the use of a single block. Our public randomness generator PubRG achieves higher output entropy than Armknecht et al.'s public randomness generators [16,17], as φ-successive blocks include at least one block that is not controlled by any adversary.

We describe subversion-resistant strategies using SRG and PubRG for cryptographic schemes. SRG can be used to instantiate the key generation algorithm and RG in symmetric and public-key encryption [18]. Our public randomness generator PubRG can be used to generate public parameters for public-key encryption (PKE) [19]. Furthermore, our proposed PubRG is suitable for other promising scenarios such as electronic lotteries [20]. The public randomness generated by PubRG can be used to yield winning tickets. Since the output of PubRG is publicly verifiable, every player who participates in an electronic lottery can verify the winning tickets.

The main contributions of this paper are summarized as follows.

• We design a public randomness generator PubRG based on public blockchains. PubRG employs hashes of blocks that are newly confirmed on a blockchain to create public randomness. It can defend against subliminal channel attacks. PubRG is also suitable for generating public parameters. The resulting randomness and parameters can accomplish public verifiability.

• We propose a secure randomness generation algorithm called SRG by instantiating double-splitting. SRG relies on two independent generators, i.e., an ordinary RG and PubRG, as well as an immunization

function to produce random coins. SRG with PubRG is secure against subliminal channel attacks.

• We provide a security analysis to demonstrate that SRG and PubRG are resistant to subliminal channel attacks in the random oracle model. We also show countermeasure strategies to thwart subliminal channel attacks for symmetric and public-key encryption.

The remainder of this paper is organized as follows. Section 2 presents the related work. In Sections 3 and 4, we present preliminaries and define the security model, respectively. In Section 5, we propose SRG. We analyze the security of SRG in Section 6. In Section 7, we discuss subversion-resistant strategies using SRG for symmetric and public-key encryption. Finally, Section 8 presents the conclusion.

2 Related work

The possibility that the implementation of cryptographic primitive deviates from the specification was first highlighted by Young and Yung [1,2] in kleptography. Since the Snowden revelations of kleptographic attacks [3], researchers have extensively investigated kleptographic attacks, including subliminal channel attacks and relevant countermeasures.

Subliminal channel attacks could compromise secrets through rejection-sampling [5]. Bellare et al. [4] observed that symmetric encryption yielding unique ciphertexts can effectively defend against such attacks on the condition that subverted ciphertexts are decryptable. Bellare et al. [21] proposed unique-ciphertext PKE. Ateniese et al. [22] extended their results and showed that the digital signature producing a unique signature is subversion-resistant, assuming subverted signatures are valid. Although the deterministic algorithms with unique output have favorable properties, probabilistic algorithms are definitely needed to achieve security such as IND-CPA security.

Mironov et al. [23] presented a cryptographic reverse firewall (that is an independent defense component) to relax the deterministic structure of the underlying algorithms to a "re-randomized" structure [24,25]. Relying on a source of trusted randomness, the reverse firewall could honestly re-randomize all inputs and outputs generated by potentially subverted randomized algorithms, thereby achieving security of subversion-resistance and security defined by specifications [26]. Analogous to cryptographic reverse firewalls, Fischlin et al. [27] proposed a self-guarding strategy that requires an honest initialization phase to thwart malicious tampering. In this phase, unbiased samples can be collected from genuine implementations. If an implementation is subverted, the unbiased samples are leveraged to re-randomize tampered outputs to prevent secret leakage. However, this approach fails to guarantee security during the initialization phase.

As mentioned above, the work in [4] requires a decryptability condition, i.e., ciphertexts output by the subverted implementation of an encryption algorithm must be correctly decrypted using the corresponding specification. Otherwise, the subversion will be simply detected. Degabriele et al. [28] relaxed the decryptability condition by considering input-trigger attacks. Such attacks render the detection probability negligible, even if decryptability is not met. Observe that decryptability can be merely satisfied via a particular form of detection, which is performed by watchdogs.

Watchdogs can individually test the functional components of algorithms to examine their veracity. Russell et al. [8] conceived subversion-resistant (trapdoor) one way permutations in the watchdog model, where a hash function can render the misbehavior of an adversary ineffective. This strategy is also referred to as the splitting mechanism. To withstand subliminal channel attacks, they further proposed double-splitting [9], where randomness generation is accomplished by mixing the outputs of two independent sources of randomness using an immunization function. With double-splitting, subversion-resistant symmetric and public-key encryption can be constructed by immunizing each component. Recently, Chow et al. [6] constructed secure digital signature schemes in the presence of kleptographic attacks. This construction is the first work on subversion-resistant signature schemes that rely merely on offline watchdogs, where the generation of keys and randomness is instantiated using double-splitting. Furthermore, to instantiate hash functions (modeled as random oracles), Chow et al. [6] applied the work of Russell et al. [29], which presented a construction that can fix a subverted random oracle such that the resulting function is "as good as" an ideal random function. Chen et al. [25] discussed how to employ double-splitting to build key encapsulation mechanisms that are secure against kleptographic attacks in the offline watchdog model. Bemmann et al. [7] designed subversion-resilient PKE with practical watchdogs that conduct constant-time tests. Additionally, Ateniese et al. [30] developed a subversion-secure immunizer in the plain model for deterministic primitives by relying on an independent source of public

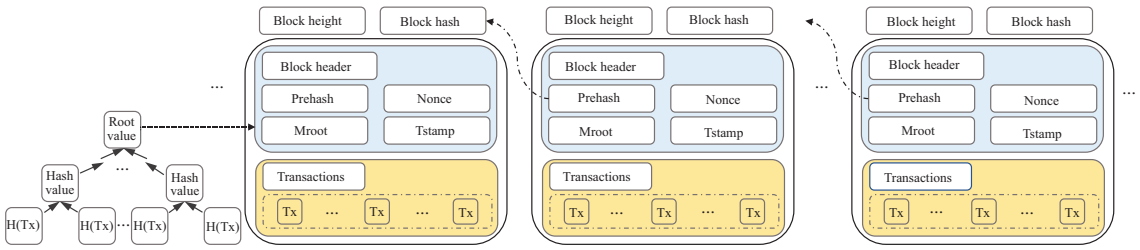

Figure 1 (Color online) Simplified Ethereum blockchain.

randomness.

On the other hand, Bonneau et al. [31] first proposed the idea of using the upcoming block on Bitcoin as a public random source. As each block on Bitcoin contains a solution to a proof-of-work puzzle, a block's min-entropy is, at minimum, equal to the difficulty of the puzzle, or else a shortcut to solving the puzzle would exist. Bonneau et al. [31] analyzed the security of this technique by defining a bribing model, in which attackers with infinite budgets might incentivize miners to mine blocks that satisfy the attackers' requirements. The attackers would suffer a monetary penalty because undesired valid blocks would be discarded and the corresponding block rewards would be forgone. Similar to the research of Bonneau et al. [31], our work relies on public blockchains such as Bitcoin. We use public blockchains as random sources and propose PubRG that is secure against kleptographic attacks. Moreover, PubRG serves as a building block for instantiating double-splitting. Additionally, we analyze the security of our technique in the watchdog model.

3 Preliminaries

3.1 Notations

We denote the set of finite bit-strings by $\{0,1\}^*$ and the set of bit-strings of length l by $\{0,1\}^l$. For a bit-string $w \in \{0,1\}^*$, $|w|$ denotes its length. Given two bit-strings x and y, we denote their concatenation by $x||y$. If Q is a set, then $|Q|$ represents the number of elements in Q. For the two sets Q and T, $Q \cup T$ denotes their union. $\lceil n \rceil$ denotes the smallest integer larger than n. We denote by $\Pr[e]$ the probability that the event e occurs. For an algorithm G, we use $y \leftarrow \mathsf{G}(x)$ to represent a run of G on input x and output y.

3.2 Blockchain

Blockchains can be viewed as distributed databases without a central authority. They are publicly verifiable and intrinsically resistant to modification [32]. A blockchain consists of a series of blocks that are linked in chronological order to form a chain and are secured by adopting a cryptographic hash function. Based on a predetermined consensus algorithm such as proof-of-work [10, 11] and proof-of-stake [33], the chain is built by a group of participants called miners. Miners append new blocks to the chain securely without needing to trust each other. When a sufficient number of blocks is added on top of a particular block, the block is confirmed and stored permanently on the blockchain.

In general, we classify blockchains into private blockchains (including consortium blockchains) and public ones. A private blockchain requires participants to be authorized by the blockchain manager, while in a public blockchain anyone can join freely. At present, public blockchains play a key role in decentralized cryptocurrencies such as Bitcoin [10] and Ethereum [11]. The blockchains serve as global ledgers that record transactions between two entities.

A simplified Ethereum is shown in Figure 1. Each block is composed of a block header and transaction data. The block header is used to calculate the current block hash, and contains a solution of proof-of-work puzzles Nonce, the hash of the last block Prehash, a timestamp Tstamp indicating when the corresponding block is mined, and a root value of a Merkle hash tree Mroot based on the transaction data. Transaction data contain all the transactions in the current block. More technical details can be found in [10, 11].

Blockchains have three properties [12–14, 34]:

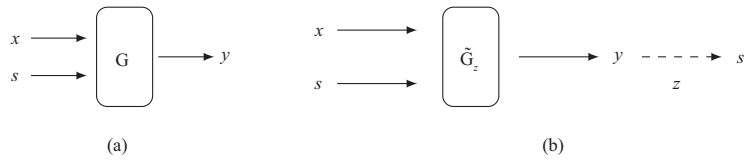

Figure 2 Embedding a subliminal channel in a randomized algorithm G. (a) An honest algorithm; (b) a subverted algorithm.

• k-chain consistency. Blockchains owned by two honest entities at any time are the same except for the trailing k blocks.

• (ι, φ)-chain quality. In any φ-consecutive blocks, at least ι of them are mined by honest parties.

• Chain growth. The blockchain owned by honest entities grows at a steady rate.

4 Security model

4.1 Subliminal channels

Subliminal channels, also known as steganographic channels, can be embedded in the outputs of subverted randomized algorithms. Through subliminal channels, secret information can be leaked to adversaries from the outputs in a way that is undetectable to other parties. Figure 2 illustrates a subliminal channel that is injected into a randomized algorithm. For an honest randomized algorithm $\mathsf{G}(x, s) := y$, where x is an input, s a secret, and y the output, a subverted algorithm $\tilde{\mathsf{G}}_z(x, s) := \tilde{y}$ produces an output $\tilde{y}$ using a hidden random string z (i.e., backdoor). $\tilde{y}$ reveals no information about s without z, since $\tilde{y}$ is indistinguishable from the output produced by $\mathsf{G}(x, s)$. However, the adversary owning z can recover all or partial bits of s from $\tilde{y}$. This illegal behavior can be thought of as a kleptographic attack, thus subliminal channels are also called subliminal channel attacks.

4.2 Stego-freeness

Stego-freeness was introduced in [9] to define the security against subliminal channel attacks. For a (randomized) algorithm G with official specification $\mathsf{G}_{\mathsf{SPEC}}$, the subverted implementation (i.e., the implementation deviating from $\mathsf{G}_{\mathsf{SPEC}}$) provided by adversary $\mathcal{A}$ is denoted by $\mathsf{G}_{\mathsf{IMPL}}$. Stego-freeness means that $\mathcal{A}$ cannot learn any more information from $\mathsf{G}_{\mathsf{IMPL}}$; otherwise, the subversion can be detected by a trusted third party referred to as watchdog $\mathcal{W}$. The watchdog $\mathcal{W}$ can interrogate $\mathsf{G}_{\mathsf{IMPL}}$ via oracle accesses to check whether it agrees with $\mathsf{G}_{\mathsf{SPEC}}$.

We can classify watchdogs into three categories. Offline watchdogs, the most practical detectors, perform a one-time check of $\mathsf{G}_{\mathsf{IMPL}}$ with oracle access. While offline watchdogs have no access to the communication transcripts between challenger $\mathcal{C}$ and adversary $\mathcal{A}$, online watchdogs can monitor and collect the transcripts by piggybacking on implementations. The most rigorous watchdogs namely omniscient watchdogs are privy to the private state (e.g., secret keys) of $\mathcal{C}$. However, instantiating online or omniscient watchdogs is costly since they need to monitor all communications actively. In this paper, we consider the offline watchdogs.

The randomized algorithm G has either a single input (security parameter 1^λ), such as the randomness generation and the key generation, or more inputs to capture the remaining cases. Without loss of generality, we assume someone (who may be the adversary) supplies a randomized input generator IG, which takes 1^λ as input and produces random outputs of length λ, to provide additional inputs for G. Next, we recall the formal definition of stego-freeness presented in [9].

Definition 1 ([9])**.** For a (randomized) algorithm G with the specification $\mathsf{G}_{\mathsf{SPEC}}$ and the potentially subverted implementation $\mathsf{G}_{\mathsf{IMPL}}$, $\mathsf{G}_{\mathsf{SPEC}}$ is stego-free in the offline-watchdog model if there exists a probabilistic polynomial-time (PPT) watchdog $\mathcal{W}$, s.t., for any PPT adversary $\mathcal{A}$ playing the security game in Figure 3, either the advantage $\mathbf{Adv}_{\mathcal{A}}(1^\lambda)$ of $\mathcal{A}$ is negligible, or the detection probability $\mathbf{Det}_{\mathcal{W},\mathcal{A}}(1^\lambda)$ of $\mathcal{W}$ is non-negligible, where λ is the security parameter. Here, $\mathbf{Adv}_{\mathcal{A}}(1^\lambda) = |\Pr[b_C = 1] - 1/2|$ and $\mathbf{Det}_{\mathcal{W},\mathcal{A}}(1^\lambda) = |\Pr[\mathcal{W}^{\mathsf{G}_{\mathsf{IMPL}}}(1^\lambda) = 1] - \Pr[\mathcal{W}^{\mathsf{G}_{\mathsf{SPEC}}}(1^\lambda) = 1]|$.

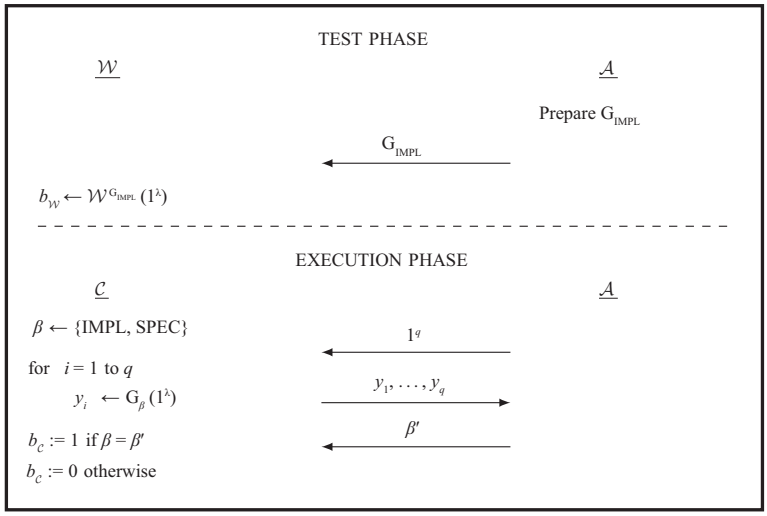

Figure 3 Stego-freeness game in the offline-watchdog model.

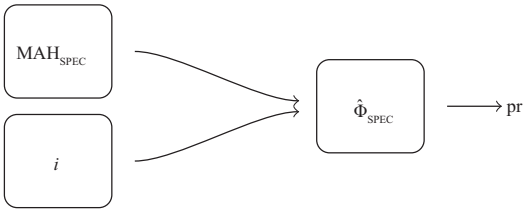

Figure 4 Simplified version of the public randomness generator specification $\mathsf{PubRG}_{\mathsf{SPEC}}$.

5 Proposed SRG

In this section, we first design a building block, namely a public randomness generator PubRG. We then construct a secure randomness generation algorithm dubbed SRG that leverages two generators (i.e., PubRG and an ordinary RG) with independence.

5.1 Public randomness generator

A simplified version of the public randomness generator specification $\mathsf{PubRG}_{\mathsf{SPEC}}$ is shown in Figure 4, where $\mathsf{MAH}_{\mathsf{SPEC}}$ denotes the specification of the functional modular of acquiring hashes (MAH), i an index, $\hat{\Phi}_{\mathsf{SPEC}}$ the specification of the immunization function $\hat{\Phi}_{\mathsf{SPEC}} : \{0,1\}^* \to \{0,1\}^l$, and pr the output. Given 1^λ, $\mathsf{PubRG}_{\mathsf{SPEC}}$ then produces uniformly random strings of length λ. Specifically, $\mathsf{MAH}_{\mathsf{SPEC}}$ acquires the hashes of φ-successive blocks that are newly confirmed on a public blockchain, such as Bitcoin and Ethereum. We denote these hashes by $\mathrm{Bl}_{h-\varphi+1}, \mathrm{Bl}_{h-\varphi+2}, \ldots, \mathrm{Bl}_h$, where h is the height of the block that is newly confirmed on the blockchain, and $\varphi \geqslant 6$ and $\varphi \geqslant 12$ are required for Bitcoin and Ethereum, respectively, due to the property of (ι, φ)-chain quality [15]. Let $k = \lceil \frac{\lambda}{l} \rceil$, after initializing the index i to zero, $\mathsf{PubRG}_{\mathsf{SPEC}}$ computes the randomness candidate $\mathrm{pr}_i = \hat{\Phi}_{\mathsf{SPEC}}(\mathrm{Bl}_{h-\varphi+1}||\mathrm{Bl}_{h-\varphi+2}||\cdots||\mathrm{Bl}_h||i+1)||\cdots||\hat{\Phi}_{\mathsf{SPEC}}(\mathrm{Bl}_{h-\varphi+1}||\mathrm{Bl}_{h-\varphi+2}||\cdots||\mathrm{Bl}_h||i+k) \bmod 2^\lambda$ using the hashes and the index. Such a strategy achieves uniformity of public randomness in the space $\{0,1\}^\lambda$. If the candidate fails to meet the predefined condition (e.g., passing the primality test for prime numbers), it needs to be recomputed with $i+k$. This process will be repeated until the condition is satisfied. Finally, $\mathsf{PubRG}_{\mathsf{SPEC}}$ outputs the parameter $\mathrm{pr} = \mathrm{pr}_i$, and publishes the relevant index i and block height h for verification. In Appendix A, we give an example to demonstrate how to use PubRG in the generation of primes.

Theorem 1. Outputs of the public randomness generator PubRG are unpredictable, publicly verifiable, and fresh.

Proof. The outputs of MAH are unpredictable. The preimage resistance of a cryptographic hash function ensures that no entity can predetermine the block hashes. Specifically, as shown in Figure 1, the block hash $\mathrm{Bl}_h = H(\mathrm{Bl}_{h-1}||\mathrm{Nonce}||\mathrm{Tstamp}||\mathrm{Mroot})$, where $H(\cdot)$ is a cryptographic hash function with

preimage resistance [35] (i.e., given a uniform y, a PPT adversary cannot find a value x satisfying $H(x) = y$). If the block hash Bl_h can be predetermined, the preimage resistance of $H(\cdot)$ is broken. Therefore, the adversary can only attempt to adjust the blocks used to derive random numbers. Nevertheless, due to the property of (ι, φ)-chain quality [12], an adversary whose hashrate is less than 51% of the network's mining hashrate cannot fully control the φ-successive blocks, and hence cannot predict the final outputs of PubRG. Furthermore, recent studies [16, 17, 31] show that a block on a public blockchain contains 64-bit min-entropy. In PubRG, we use φ blocks to obtain high entropy, which meets the requirement of randomness.

We stress that the unpredictability of block hashes produced in the future does not mean that the hashes cannot be biased by an adversary who has an infinite budget. The adversary might incentivize a miner who first mines a block to discard the newly mined block and continue to mine if the block hash does not meet the requirement of the adversary. However, this is very costly as the adversary must simultaneously perform a huge number of calculations to find valid blocks and forgo the block rewards [31]. More importantly, the final block hash is still unpredictable to adversaries.

The immunization function $\hat{\Phi}$ (i.e., hash function) is modeled as a random oracle. The output distribution of PubRG is computationally indistinguishable from a uniform distribution in the presence of PPT adversaries. Additionally, due to the collision resistance of the hash function, any change of the input will lead to a different output. For the honest function $\hat{\Phi}$, the only way to bias its output is to determine the output in advance and adjust the input. However, this is impossible as the input (i.e., the output of MAH) is unpredictable.

As for public verifiability, according to the block height h issued, anyone is able to acquire the same hashes $\mathrm{Bl}_{h-\varphi+1}, \mathrm{Bl}_{h-\varphi+2}, \ldots, \mathrm{Bl}_h$ from the public blockchain due to the blockchains' chain consistency property [14]. Once the hashes are determined, there will be an index i' satisfying the specified condition exactly. A verifier can first search for i' and check whether i' matches with the published one. Then, it verifies the correctness of pr by checking $\mathrm{pr} \stackrel{?}{=} \hat{\Phi}_{\mathsf{SPEC}}(\mathrm{Bl}_{h-\varphi+1}||\mathrm{Bl}_{h-\varphi+2}||\cdots||\mathrm{Bl}_h||i+1)||\cdots||\hat{\Phi}_{\mathsf{SPEC}}(\mathrm{Bl}_{h-\varphi+1}||\mathrm{Bl}_{h-\varphi+2}||\cdots||\mathrm{Bl}_h||i+k) \bmod 2^\lambda$, where $k = \lceil \frac{\lambda}{l} \rceil$.

Since we use the hash of the newly confirmed block, the timestamp Tstamp of the block enables anyone to ensure the outputs of PubRG are newly generated, and thus, freshness is guaranteed.

Lemma 1 ([8])**.** Consider an adversarial implementation $\Pi_{\mathsf{IMPL}} := (F^1_{\mathsf{IMPL}}, \ldots, F^k_{\mathsf{IMPL}})$ of a specification $\Pi_{\mathsf{SPEC}} := (F^1_{\mathsf{SPEC}}, \ldots, F^k_{\mathsf{SPEC}})$, where $F^1_{\mathsf{IMPL}}, \ldots, F^k_{\mathsf{IMPL}}$ denote deterministic algorithms with input distributions $X^1_\lambda, \ldots, X^k_\lambda$, respectively, and λ denotes the security parameter. If $\exists j \in [k]$ such that $\Pr[F^j_{\mathsf{IMPL}}(x) \neq F^j_{\mathsf{SPEC}}(x) : x \leftarrow X^j_\lambda] = \delta$, a PPT offline watchdog can detect whether $F^j_{\mathsf{SPEC}} \stackrel{?}{=} F^j_{\mathsf{IMPL}}$ with probability at least δ.

Theorem 2. Consider a public randomness generator PubRG with the specification $(\mathsf{MAH}_{\mathsf{SPEC}}, \hat{\Phi}_{\mathsf{SPEC}}, i)$ illustrated in Figure 4:

- $\mathsf{MAH}_{\mathsf{SPEC}}$ outputs hashes of φ-successive blocks that are newly confirmed on a public blockchain;
- $\hat{\Phi}_{\mathsf{SPEC}}$ is a hash function that takes a string $w \in \{0,1\}^*$ as input and outputs a string of length l;
- i is a public index belonging to the natural number set;
- the specification for $\mathsf{PubRG}(1^\lambda)$ is a trusted amalgamation.

$\mathsf{PubRG}_{\mathsf{SPEC}}$ is stego-free with an offline watchdog if $\hat{\Phi}_{\mathsf{SPEC}}$ is modeled as a random oracle.

First, note the following two facts.

On the one hand, the offline watchdog $\mathcal{W}$ ensures that blockchains are publicly verifiable and resistant to modification due to chain consistency [14,15]. Since blockchains are public ledgers, any entity is capable of accessing them. In PubRG, the output of MAH is a set of block hashes. If the output of $\mathsf{PubRG}_{\mathsf{IMPL}}$ is not identical to the corresponding block hashes, differences between $\mathsf{MAH}_{\mathsf{IMPL}}$ and $\mathsf{MAH}_{\mathsf{SPEC}}$ can be discovered.

On the other hand, according to Lemma 1, for a deterministic algorithm implemented by adversaries, $\mathcal{W}$ can guarantee that the implementation is consistent with the specification (i.e., $\mathsf{G}_{\mathsf{SPEC}}(x) = \mathsf{G}_{\mathsf{IMPL}}(x)$) with high probability when inputs are sampled from a public input distribution. $\mathcal{W}$ evaluates $\mathsf{G}_{\mathsf{SPEC}}$ and $\mathsf{G}_{\mathsf{IMPL}}$ with input x drawn from a known distribution, and then checks whether $\mathsf{G}_{\mathsf{SPEC}}$ agrees with $\mathsf{G}_{\mathsf{IMPL}}$. Accordingly, the specification of a deterministic algorithm with a public input distribution is stego-free.

Proof. If $\mathsf{PubRG}_{\mathsf{SPEC}}$ is stego-free, then for any implementation $\mathsf{PubRG}_{\mathsf{IMPL}} := (\mathsf{MAH}_{\mathsf{IMPL}}, \hat{\Phi}_{\mathsf{IMPL}}, i)$ provided by adversary $\mathcal{A}$, $\mathcal{A}$ cannot differentiate between $\mathsf{PubRG}_{\mathsf{IMPL}}$ and $\mathsf{PubRG}_{\mathsf{SPEC}}$, or a watchdog $\mathcal{W}$ can detect the difference. Next, we will show that the detection probability $\mathbf{Det}_{\mathcal{W},\mathcal{A}}$ of $\mathcal{W}$ is non-

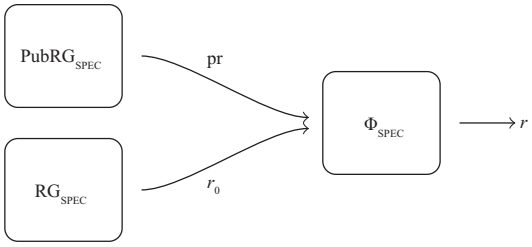

Figure 5 Specification of the secure randomness generation algorithm $\mathsf{SRG}_{\mathsf{SPEC}}$.

negligible.

Suppose $\mathcal{A}$ replaces $\hat{\Phi}_{\mathsf{SPEC}}$ with $\hat{\Phi}_{\mathsf{IMPL}}$, by Lemma 1, then $\mathbf{Det}_{\mathcal{W},\mathcal{A}} \geqslant \delta$. Since $\hat{\Phi}_{\mathsf{SPEC}}$ is a deterministic algorithm with a public input distribution (i.e., $\mathsf{MAH}_{\mathsf{IMPL}} \times i$), $\mathcal{W}$ is able to detect the inconsistency between $\hat{\Phi}_{\mathsf{IMPL}}$ and $\hat{\Phi}_{\mathsf{SPEC}}$ with probability at least δ.

For a subverted implementation where only $\mathsf{MAH}_{\mathsf{SPEC}}$ is replaced with $\mathsf{MAH}_{\mathsf{IMPL}}$, the detection probability is non-negligible (i.e., $\mathbf{Det}_{\mathcal{W},\mathcal{A}} \geqslant \delta_b$). Since blockchains are publicly verifiable and intrinsically immutable, the correctness of hashes output from $\mathsf{MAH}_{\mathsf{SPEC}}$ can be easily verified. Thus, any malicious implementation will be detected by $\mathcal{W}$.

Consequently, $\mathcal{W}$ can detect the disagreement between $\mathsf{PubRG}_{\mathsf{IMPL}}$ and $\mathsf{PubRG}_{\mathsf{SPEC}}$ with a probability of at least $\min(\delta_i, \delta_b)$.

5.2 Construction of SRG

SRG is based on an immunization function and two physically independent random sources, i.e., PubRG and an ordinary RG.

In SRG, RG is first executed to produce the randomness r_0. Then, when a new block is confirmed on the public blockchain, PubRG creates the public randomness pr. The concatenation of r_0 and pr is fed into an immunization function $\Phi : \{0,1\}^{2t} \to \{0,1\}^t$ to obtain the final output $r = \Phi(\mathrm{pr}||r_0)$. Such a mechanism defends against attempts by RG to launch rejection-sampling, since the generation of r_0 cannot depend on pr, which is yielded later. On the other hand, the generation of pr cannot be controlled by adversaries according to Theorem 2. Consequently, SRG cannot evaluate $\Phi(\mathrm{pr}||r_0)$ during the generation of pr or r_0 since the two randomness generators are independent. If Φ is modeled as a random oracle, then r is independent of the backdoor of the adversary $\mathcal{A}$. The subliminal channel attacks are resisted. Figure 5 shows the specification of SRG.

6 Security analysis

In this section, we prove that our proposed secure randomness generator SRG's specification ($\mathsf{SRG}_{\mathsf{SPEC}}$) is stego-free in the random oracle model. We then demonstrate that the specification of a randomized algorithm with a public input distribution is stego-free when adopting $\mathsf{SRG}_{\mathsf{SPEC}}$.

Theorem 3. Consider a secure randomness generation algorithm SRG with the specification ($\mathsf{PubRG}_{\mathsf{SPEC}}$, $\mathsf{RG}_{\mathsf{SPEC}}, \Phi_{\mathsf{SPEC}}$) described below:

- $\mathsf{PubRG}_{\mathsf{SPEC}}$, given 1^λ, outputs uniformly random strings of length λ;
- Given 1^λ, $\mathsf{RG}_{\mathsf{SPEC}}$ outputs uniformly random strings of length λ;
- Φ_{SPEC} is a hash function and $\Phi_{\mathsf{SPEC}}(w)$ has length $\lceil |w|/2 \rceil$;
- The specification for $\mathsf{SRG}(1^\lambda)$ is a trusted composition: $\Phi_{\mathsf{SPEC}}(\mathsf{PubRG}_{\mathsf{SPEC}}(1^\lambda), \mathsf{RG}_{\mathsf{SPEC}}(1^\lambda))$.

$\mathsf{SRG}_{\mathsf{SPEC}}$ is stego-free with an offline watchdog if Φ_{SPEC} is modeled as a random oracle.

Proof. SRG is an instantiation of double-splitting, whose proof is based on two basic guarantees provided by an offline watchdog $\mathcal{W}$ [9]. First, $\mathcal{W}$ ensures that the discrepancy between the implementation of a deterministic algorithm with a public input distribution and the corresponding specification can be detected with non-negligible probability, as shown in Lemma 1. Second, $\mathcal{W}$ guarantees that outputs of a randomness generator's implementation are unpredictable to adversaries. Note that $\mathcal{W}$ also provides the two guarantees to prove this theorem. In the second guarantee, the implementations of both randomness generators in SRG, namely PubRG and RG, have unpredictability. The output of $\mathsf{RG}_{\mathsf{IMPL}}$ cannot be predicted by adversaries; otherwise, the output distribution of $\mathsf{RG}_{\mathsf{IMPL}}$ would incur collision with a

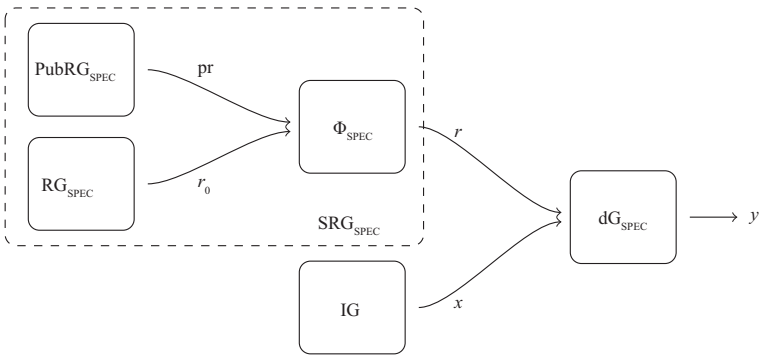

Figure 6 Stego-free specification for randomized algorithm G, where $x \leftarrow \mathsf{IG}$.

significant probability [9]. Besides, according to Theorem 1, the output of $\mathsf{PubRG}_{\mathsf{IMPL}}$ is unpredictable to adversaries. The proof of this theorem is analogous to the stego-freeness proof of double-splitting, i.e., Theorem 3.4 in [9]. Hence, we omit the concrete proof.

Theorem 4. The specification $\mathsf{G}_{\mathsf{SPEC}} := (\mathsf{SRG}_{\mathsf{SPEC}}, \mathsf{dG}_{\mathsf{SPEC}})$ for any randomized algorithm G is illustrated in Figure 6. Given 1^λ, $\mathsf{SRG}_{\mathsf{SPEC}}$ with the form $(\mathsf{PubRG}_{\mathsf{SPEC}}, \mathsf{RG}_{\mathsf{SPEC}}, \Phi_{\mathsf{SPEC}})$ outputs uniformly random strings of length λ (see Figure 5). $\mathsf{dG}_{\mathsf{SPEC}}$ is a deterministic algorithm that takes the outputs of $\mathsf{SRG}_{\mathsf{SPEC}} \times \mathsf{IG}$ as its input, where IG is an input generator to provide an extra input for G. When the amalgamation is trusted, $\mathsf{G}_{\mathsf{SPEC}}$ is stego-free with an offline watchdog if Φ_{SPEC} is modeled as a random oracle.

The watchdog $\mathcal{W}$ to detect $\mathsf{G}_{\mathsf{SPEC}} := (\mathsf{SRG}_{\mathsf{SPEC}}, \mathsf{dG}_{\mathsf{SPEC}})$ is a combination of two watchdogs. One watchdog guarantees the stego-freeness of $\mathsf{SRG}_{\mathsf{SPEC}}$ according to Theorem 3. The other watchdog makes sure, by Lemma 1, that $\mathsf{dG}_{\mathsf{IMPL}}$ agrees with $\mathsf{dG}_{\mathsf{SPEC}}$ on the inputs sampled from a public distribution $\mathsf{SRG}_{\mathsf{SPEC}} \times \mathsf{IG}$. Hence, the proof is omitted.

7 Subversion-resistant strategies for cryptographic schemes

In this section, we present subversion-resistant strategies for symmetric encryption and PKE using SRG.

7.1 Subversion-resistant symmetric encryption

A symmetric encryption scheme consists of a key generator KG, an encryption algorithm Enc, and a decryption algorithm Dec, where Enc can be further split into a randomness generator RG' and a deterministic algorithm dEnc. By Lemma 1, the deterministic algorithm specification $\mathsf{Dec}_{\mathsf{SPEC}}$ is stego-free. We can use $\mathsf{SRG}_{\mathsf{SPEC}}$ to serve as $\mathsf{KG}_{\mathsf{SPEC}}$ and $\mathsf{RG}'_{\mathsf{SPEC}}$ to guarantee stego-freeness. $\mathsf{Enc}_{\mathsf{SPEC}}$ is stego-free according to Theorem 4. Figure 7 shows the stego-free specification for symmetric encryption.

7.2 Subversion-resistant public-key encryption

The construction of subversion-resistant PKE follows from that of subversion-resistant symmetric encryption except for two differences. First, the subversion-resistance of public parameters (e.g., primes) needs to be considered in PKE. The existing studies [1, 2] describe a secretly embedded trapdoor with universal protection (SETUP) attack to embed backdoors into public parameters. Adversaries owning the backdoors may obtain secret information (e.g., secret keys) from the public parameters inconspicuously. We can use PubRG to produce public parameters that are verifiable and secure against SETUP (see Theorems 1 and 2).

Second, the key generation of PKE is more complex, since the asymmetric keys consist of a public key pk and a private key sk. Compared with the subversion-resistant symmetric encryption, the specification of asymmetric key generation algorithm $\mathsf{KG}_{\mathsf{SPEC}}$ has an extra element, namely a deterministic algorithm specification $\mathsf{dKG}_{\mathsf{SPEC}}$. Figure 8 shows a stego-free specification for encryption in PKE, where the randomness generator and the key generation algorithm are instantiated via SRG.

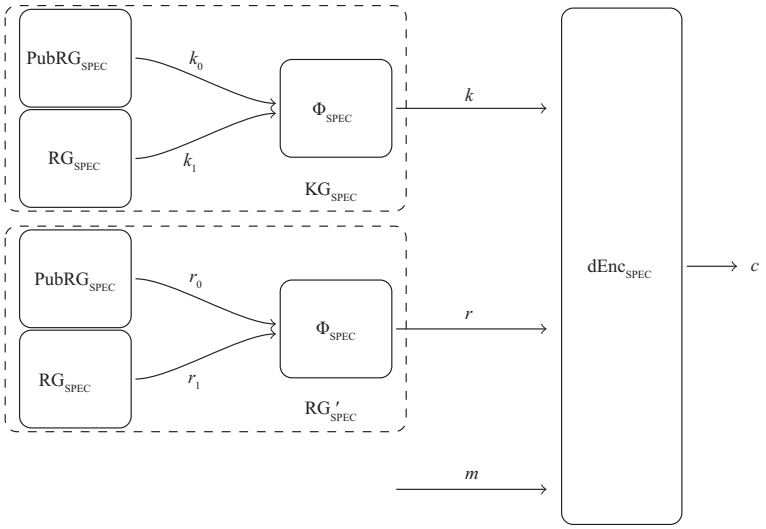

Figure 7 Stego-free specification for symmetric encryption.

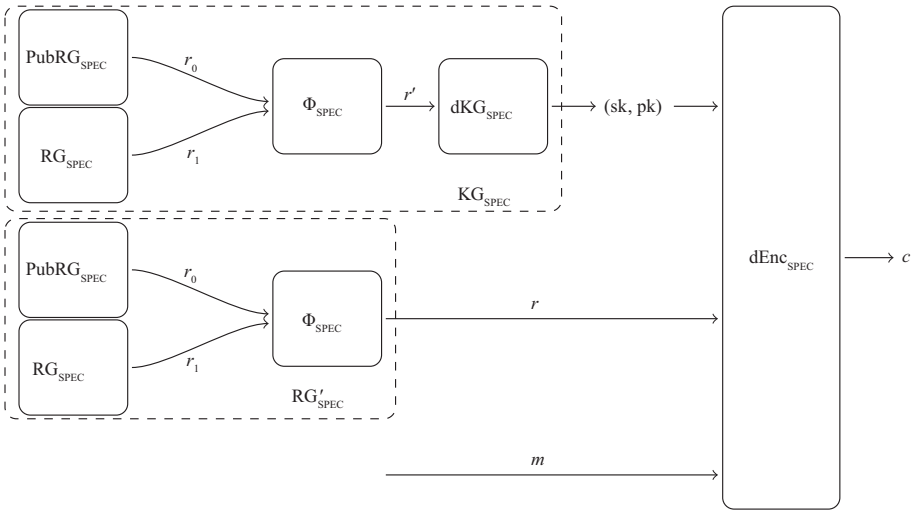

Figure 8 Stego-free specification for encryption in PKE.

8 Conclusion

In this paper, we proposed a secure randomness generation algorithm dubbed SRG by instantiating double-splitting. SRG relies on two independent randomness generators, and we designed a public randomness generator PubRG based on public blockchains as one of the two generators. Security analysis shows that SRG and PubRG can achieve immunization against subliminal channel attacks. Using SRG and PubRG, we described countermeasure strategies to resist subliminal channel attacks for symmetric and public-key encryption.

Acknowledgements This work was supported in part by National Nature Science Foundation of China (Grant Nos. 62272091, 61872060) and National Key R&D Program of China (Grant No. 2017YFB0802000).

References

1 Young A, Yung M. The dark side of "black-box" cryptography or: should we trust capstone? In: Proceedings of Annual International Cryptology Conference, 1996. 89–103
2 Young A, Yung M. Kleptography: using cryptography against cryptography. In: Proceedings of International Conference on the Theory and Applications of Cryptographic Techniques, 1997. 62–74
3 Perlroth N, Larson J, Shane S. NSA able to foil basic safeguards of privacy on web. The New York Times, 2013. http://www.nytimes.com/2013/09/06/us/nsa-foils-much-internet-encryption.html
4 Bellare M, Paterson K G, Rogaway P. Security of symmetric encryption against mass surveillance. In: Proceedings of Annual Cryptology Conference, 2014. 1–19
5 Bellare M, Jaeger J, Kane D. Mass-surveillance without the state: strongly undetectable algorithm-substitution attacks. In: Proceedings of the 22nd ACM Conference on Computer and Communications Security, 2015. 1431–1440

6 Chow S S, Russell A, Tang Q, et al. Let a non-barking watchdog bite: cliptographic signatures with an offline watchdog. In: Proceedings of IACR International Workshop on Public Key Cryptography, 2019. 221–251
7 Bemmann P, Chen R, Jager T. Subversion-resilient public key encryption with practical watchdogs. In: Proceedings of IACR International Conference on Public-Key Cryptography, 2021. 627–658
8 Russell A, Tang Q, Yung M, et al. Cliptography: clipping the power of kleptographic attacks. In: Proceedings of International Conference on the Theory and Application of Cryptology and Information Security, 2016. 34–64
9 Russell A, Tang Q, Yung M, et al. Generic semantic security against a kleptographic adversary. In: Proceedings of ACM SIGSAC Conference on Computer and Communications Security, 2017. 907–922
10 Nakamoto S. Bitcoin: a peer-to-peer electronic cash system. 2008. https://bitcoin.org/bitcoin.pdf
11 Wood G. Ethereum: a secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper, 2014, 151: 1–32
12 Garay J, Kiayias A, Leonardos N. The Bitcoin backbone protocol: analysis and applications. In: Proceedings of Annual International Conference on the Theory and Applications of Cryptographic Techniques, 2015. 281–310
13 Badertscher C, Maurer U, Tschudi D, et al. Bitcoin as a transaction ledger: a composable treatment. In: Proceedings of Annual International Cryptology Conference, 2017. 324–356
14 Pass R, Seeman L, Shelat A. Analysis of the blockchain protocol in asynchronous networks. In: Proceedings of Annual International Conference on the Theory and Applications of Cryptographic Techniques, 2017. 643–673
15 Zhang Y, Xu C, Cheng N, et al. Chronos^{+}: an accurate blockchain-based time-stamping scheme for cloud storage. IEEE Trans Serv Comput, 2020, 13: 216–229
16 Armknecht F, Bohli J M, Karame G O, et al. Transparent data deduplication in the cloud. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, 2015. 886–900
17 Armknecht F, Bohli J M, Karame G O, et al. Outsourced proofs of retrievability. In: Proceedings of ACM SIGSAC Conference on Computer and Communications Security, 2014. 831–843
18 Sun L X, Xu C X, Zhang M W, et al. Secure searchable public key encryption against insider keyword guessing attacks from indistinguishability obfuscation. Sci China Inf Sci, 2018, 61: 038106
19 Jiang C, Xu C, Cao C, et al. GAIN: decentralized privacy-preserving federated learning. J Inf Secur Appl, 2023, 78: 103615
20 Chow S S M, Hui L C K, Yiu S M, et al. Practical electronic lotteries with offline TTP. Comput Commun, 2006, 29: 2830–2840
21 Bellare M, Hoang V T. Resisting randomness subversion: fast deterministic and hedged public-key encryption in the standard model. In: Proceedings of Annual International Conference on the Theory and Applications of Cryptographic Techniques, 2015. 627–656
22 Ateniese G, Magri B, Venturi D. Subversion-resilient signature schemes. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, 2015. 364–375
23 Mironov I, Stephens-Davidowitz N. Cryptographic reverse firewalls. In: Proceedings of Annual International Conference on the Theory and Applications of Cryptographic Techniques, 2015. 657–686
24 Chen R, Mu Y, Yang G, et al. Cryptographic reverse firewall via malleable smooth projective hash functions. In: Proceedings of International Conference on the Theory and Application of Cryptology and Information Security, 2016. 844–876
25 Chen R, Huang X, Yung M. Subvert KEM to break DEM: practical algorithm-substitution attacks on public-key encryption. In: Proceedings of International Conference on the Theory and Application of Cryptology and Information Security, 2020. 98–128
26 Jiang C, Xu C, Zhang Z, et al. SR-PEKS: subversion-resistant public key encryption with keyword search. IEEE Trans Cloud Comput, 2023, 11: 3168–3183
27 Fischlin M Mazaheri S. Self-guarding cryptographic protocols against algorithm substitution attacks. In: Proceedings of the 31st IEEE Computer Security Foundations Symposium, 2018. 76–90
28 Degabriele J P, Farshim P, Poettering B. A more cautious approach to security against mass surveillance. In: Proceedings of International Workshop on Fast Software Encryption, 2015. 579–598
29 Russell A, Tang Q, Yung M, et al. Correcting subverted random oracles. In: Proceedings of Annual International Cryptology Conference, 2018. 241–271
30 Ateniese G, Francati D, Magri B, et al. Public immunization against complete subversion without random oracles. In: Proceedings of International Conference on Applied Cryptography and Network Security, 2019. 465–485
31 Bonneau J, Clark J, Goldfeder S. On Bitcoin as a public randomness source. 2015. https://eprint.iacr.org/2015/1015.pdf
32 Jiang C, Xu C, Zhang Y. PFLM: privacy-preserving federated learning with membership proof. Inf Sci, 2021, 576: 288–311
33 Kiayias A, Russell A, David B, et al. Ouroboros: a provably secure proof-of-stake blockchain protocol. In: Proceedings of Annual International Cryptology Conference, 2017. 357–388
34 Kiayias A, Panagiotakos G. Speed-security tradeoffs in blockchain protocols. 2015. https://eprint.iacr.org/2015/1019.pdf
35 Katz J, Lindell Y. Introduction to Modern Cryptography. Boca Raton: CRC Press, 2020

Appendix A An example of using PubRG

Assume that we generate a 32-bit prime number p with the PubRG, where Ethereum serves as the public blockchain and SHA-256 the immunization function. First, the functional modular of acquiring hashes (MAH) gets the hashes of 12 blocks that are newly confirmed on Ethereum (at 2022-01-11 06:13:14 +UTC). These hashes and relevant block heights are shown in Table A1. Let $\mathrm{Bl}_{\mathrm{height}}$ denote the hash of the block whose height is height. In this example, $l = 256$ and $\lambda = 32$, so that $k = \lceil \frac{\lambda}{l} \rceil = 1$. After initializing the index i to be 0, PubRG can compute a prime candidate $\mathrm{pr}_0 = \text{SHA-256}(\mathrm{Bl}_{13982511}||\mathrm{Bl}_{13982512}||\cdots||\mathrm{Bl}_{13982522}||1) \bmod 2^{32} = 530181258$. Since pr_0 cannot pass the primality test, PubRG needs to recompute a prime candidate with $i = 1$ to obtain $\mathrm{pr}_1 = \text{SHA-256}(\mathrm{Bl}_{13982511}||\mathrm{Bl}_{13982512}||\cdots||\mathrm{Bl}_{13982522}||2) \bmod 2^{32} = 494169299$. pr_1 is not a prime either. A new candidate will be calculated with $i = 2$. The process is repeated until $i = 11$, and a prime candidate $\mathrm{pr}_{11} = \text{SHA-256}(\mathrm{Bl}_{13982511}||\mathrm{Bl}_{13982512}||\cdots||\mathrm{Bl}_{13982522}||12) \bmod 2^{32} = 2138032957$ survives the primality test. Finally, PubRG outputs the 32-bit prime number $p = \mathrm{pr}_{11}$, and publishes the relevant index $i = 11$ and block height $h = 13982522$ for verification.

Table A1 Hashes of the newly confirmed blocks

Block height	Block hash
13982511	0x28aa67052049c03d57dc69272e672270cecc283bced965c0662263977128ee81
13982512	0x34db27c61773f9a7e165b429339c0b26fdffd8cbc652e3ddcc5b07ad1b888594
13982513	0x33cd0a433fcb7ca0af5ede95d02ed1d1be498b4639df5a6831aeb35cc08ea9a2
13982514	0x3f13006f208fa1dbb469a755fc59defc8f4d528f89f3e69c76b5c06cf98b83a9
13982515	0x25d4d50cfd6da080eab633ab63d16bb1c2cc40bf0a092d9ac1dcecdd64a8105d
13982516	0x2e489d63b18b1f6c33e616aa45d10c19b26b270d851a3e57c0cc9a4e38fa1f79
13982517	0x6621e8edf3054073e71b76c462c9eb03e4e63b947adba0dc657c50bd0224bbf6
13982518	0x5caee2b00a0c2ef05861fd24d5f1e0424698873b621240cced34025a0f0db075
13982519	0x60d81afe935def5de75c0ac7b9ddb7dbd5f7c6b81c8ed84fa413e084d6db1882
13982520	0xab122933f55657cc966a1ac4f1e4a2b65cf56708c82ffab10ddd45e78ca9fcf2
13982521	0x229fdad0bc72c1d7932497b48c399b606089d3291bade0d011e150cf5fc9fde2
13982522	0x97f02dbd787cfda81818665c3617e3b3ab63891cdb564ace6d82df515c58d889