Authentic Quantum Nonces

Stefan Rass, Peter Schartner and Jasmin Wachter
Department of Applied Informatics, System Security Group
Universität Klagenfurt, Universitätsstrasse 65-67
9020 Klagenfurt, Austria
email: {stefan.rass, peter.schartner, jasmin.wachter}@aau.at

***Abstract*—Random numbers are an important ingredient in cryptographic applications, whose importance is often underestimated. For example, various protocols hinge on the requirement of using numbers only once and never again (most prominently, the one-time pad), or rest on a certain minimal entropy of a random quantity. Quantum random number generators can help fulfilling such requirements, however, they may as well be subject to attacks. Here, we consider what we coin a *randomness substitution attack*, in which the adversary replaces a good randomness source by another one, which produces duplicate values (over time) and perhaps numbers of low entropy. A binding between a random number and its origin is thus a certificate of quality and security, when upper level applications rest on the good properties of quantum randomness.**

Keywords–Quantum Cryptography; Randomness Substitution Attack; Random Number Generation; Security; Authentication.

I. Motivation

Random numbers play different roles in cryptographic systems. Mostly, they are used to generate keys or create uncertainty towards better security in different attack scenarios. Concerning the latter, it is often necessary to assure a certain minimum entropy of a random value, and to prevent coincidental equality of two random numbers chosen at different times or different places. While the former requirement is obvious, revealing the problem with the latter requires some more arguing: as a simple example, consider two independent persons A, B instantiating individual RSA (Rivest-Shamir-Adleman) encryption systems. Both choose large primes p_A, q_A and p_B, q_B, respectively, making up the key-parameters $n_A = p_A q_A$ and $n_B = p_B q_B$. If $\{p_A, q_A\} \cap \{p_B, q_B\} \neq \emptyset$ and $n_A \neq n_B$, then $\gcd(n_A, n_B) \in \{p_A, p_B, q_A, q_B\}$, which defeats security of both RSA instances. Adhering to recommended key-sizes, it is tempting to think that the chances of a match of two, say 512 bit long, primes is negligible. Even mathematically, the prime number theorem assures that there are at least 1.84×10^{151} primes within the range $\{2^{511}, \ldots, 2^{512} - 1\}$, so there appears to be no problem in choosing those parameters independently from each other. Unfortunately, reality differs from the theoretical expectations in a devastating manner: according to findings of [1], approximately 12,500 out of more than 4.7 million RSA-moduli could be factored by humble pairwise greatest common division computation!

At least for this reason, quantum randomness would – at first glance – be a good replacement for user-supplied randomness (such as mouse movements). However, a proper post-processing to authenticate a generator's output and to avoid random number generators coming up with identical outputs is nevertheless an advisable precaution.

Furthermore, while the statistical odds to accidentally hit the same integer over a search in the range of 512 bit or higher is sure negligible, reframing this possibility towards a potential attack scenario is worthwhile to look at. Especially so, as standard cryptosystems like RSA or ElGamal (and hence also the digital signature standard) can be attacked most easily, when the involved randomness source gets under the attacker's control or influence, regardless of whether or not the randomness is used to find primes or simply as a general input. We call this a *randomness substitution attack*. Scaling up this thought, distributed attacks on random number generators that make only a portion of those emit random numbers with low entropy may already suffice to establish a significant lot of RSA instances [2] that are vulnerable to simple $\gcd$-based factorization, or instances of ElGamal signatures [3] [4] (such as the digital signature standard is based on), where the secret key sk can easily be recovered if the *same* signature exponent k in $r = g^k \text{ MOD } (p-1)$ is used twice, e.g., if the random number generator has been hacked.

The foremost danger of randomness substitution is not its sophistication, but its simplicity and apparent insignificance that may cause countermeasures to be hardly considered as necessary. Nevertheless, authentic random values with lower-bounded entropy and explicit avoidance of coincidental matches are easy to construct yet advisable to use.

The paper is organized as follows. In Section II, we will sketch the basic cryptographic building blocks used to embed certain additional information into a quantum-generated random bitstring. This additional information will not only assure distinctness of values generated by otherwise independent generators, but also assure uniqueness of values over an exponentially long range in the (infinite) sequence of random numbers emitted by the same generator. We call such numbers *nonces*. Section III shows the construction and how to verify the origin of a random number. Notice that in this context, we neither claim nor demand information-theoretic security (as would be common in a full-fledged quantum cryptographic setting), but our focus is on classical applications that use quantum randomness to replace user-supplied random values. However, replacing the generator itself is an issue that must as well be avoided, which is doable by classical techniques, as we will outline here.

II. Preliminaries

Let $x \in \{0,1\}^\ell$ denote bitstrings of length ℓ, and let $\{0,1\}^*$ be the set of all bitstrings (of arbitrary length). The notation $x\|y$ denotes any encoding of x and y into a new string, from which a unique recovery of x and y is possible (e.g., concatenation of x and y, possibly using a separator symbol). Sets are written in sans serif letters, such as M, and their cardinality is $|\mathsf{M}|$.

To establish a binding betweeen random numbers and their origin devices, and to assure uniqueness of random values over time and across different number generators, we will employ digital signatures with message recovery, and symmetric encryption. Recall the general framework of these, into which RSA-, Rabin or Nyberg-Rueppel signatures fit [5]: let $\mathsf{M} \subseteq \{0,1\}^*$ be the *message space*, and let M_S be the *signing space*, i.e., the set of all (transformed) messages on which we may compute a digital signature. Furthermore, let $R : \mathsf{M} \to \mathsf{M}_S$ be an invertible *redundancy function* that is publicly known, and for simplicity, equate $R(\mathsf{M}) = \text{Im}(R) = \mathsf{M}_S$. We define the mappings $Sign : \mathsf{M}_S \times \mathsf{K} \to \mathsf{S}$ and $Extract : \mathsf{S} \times \mathsf{K} \to \text{Im}(R)$ as the signing and verification functions, where S is the signature space and K is the keyspace, where the secret signature key and public verification key come from.

A digital signature is obtained by computing $s = Sign(R(m), sk)$. As we demand message recovery, the verification proceeds in four steps, assuming that we received the signature s^* to be validated:

1) Obtain the signer's public key pk from a valid certificate (also provided by the signer),
2) Compute $\widetilde{m} = Extract(s^*)$.
3) Verify that $\widetilde{m} \in \text{Im}(R) = \mathsf{M}_S$, otherwise reject the signature.
4) Recover the message $m = R^{-1}(\widetilde{m})$.

Our construction to follow in Section III will crucially rely on the recovery feature of the signature, so that resilience against existential forgery mostly hinges on a proper choice of the redundancy function R. In general, this choice should be made dependent on the signature scheme in charge, and to thwart existential forgery, the redundancy function should not exhibit any homomorphic properties. A possible choice would be $R(m) = m\|h(m)$, where h is a cryptographic (or universal) hash-function, where we emphasize that no rigorous security proof of this choice is provided here.

As a second ingredient, we will use a symmetric encryption E, writing $E_k(m)$ to mean the encryption of m under key k and transformation E. The respective decryption is denoted as $E_k^{-1}(m)$. Our recommended choice for practicality is the Advanced Encryption Standard (AES).

Finally, we assume that each random generator is equipped with a world-wide unique identification number, such as is common for network cards (Media-Access-Control (MAC) address) or smartcards (Integrated Circuit Card Serial Number (ICCSN) [6]). Hereafter, we will refer to this quantity as the ID of the generator.

III. Construction

Given a generator equipped with a unique identifier ID and an internal counter $c \in \mathbb{N}$ (initialized to zero), let $r \in \{0,1\}^*$ denote a raw random bitstring that the quantum random generator emits per invocation.

The final output of the random generator is now constructed over the following steps, (see Figure 1).

1) Increment $c \leftarrow c + 1$
2) Compute $x \leftarrow ID\|c\|r$
3) Apply a digital signature with message recovery, using the secret signature key sk, i.e., compute $s \leftarrow Sign(R(x), sk)$.

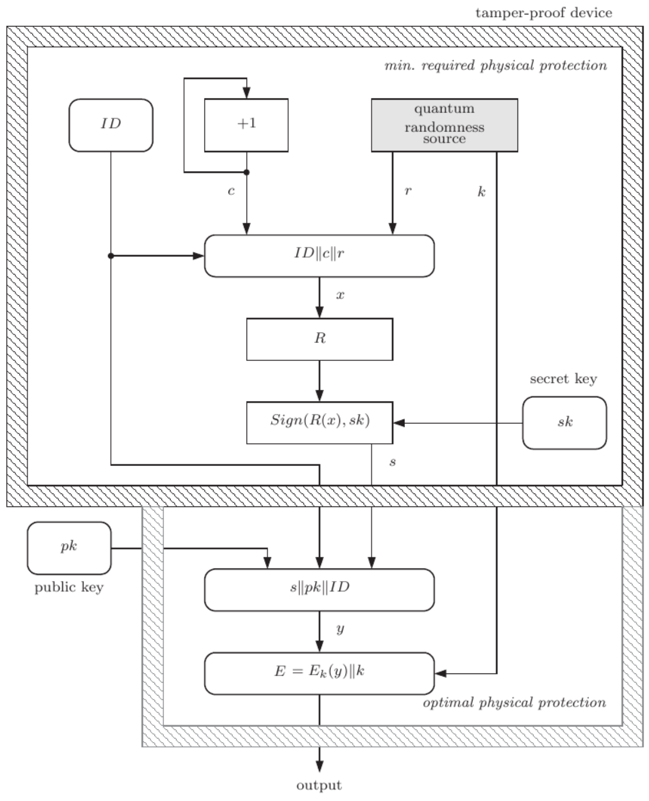

Figure 1. Schematic of post-processing for authentic nonces

4) Append the generator's public key pk and identity ID to get $y \leftarrow s\|pk\|ID$
5) Choose another (quantum) random number k and deliver the final output (the authentic nonce)

$$z := E_k(y)\|k.$$

It is easy to see that the so-constructed sequence of numbers enjoys all the properties that we are looking for. The last step ensures randomness, as parts of the random values (the public key and identity) remain constant over time. Note that all of the above transformations are invertible and hence injective. We examine each of the properties separately in the following.

a) Uniqueness: To this end, let $z_1 = E_{k_1}(y_1)\|k_1, z_2 = E_{k_2}(y_2)\|k_2$ be two outputs of a generator (possibly the same one or different devices). Uniqueness is trivial if $k_1 \neq k_2$, so assume a coincidental match between the two or the possibility that k_1, k_2 origin from an attacker. If z_1, z_2 match upon the least significant bits making up the keys $k_1 = k_2 = k$, then uniqueness requires $E_k(y_1) \neq E_k(y_2)$. Since E_k is injective, we hence look at $y_1 = s_1\|pk_1\|ID_1$ and $y_2 = s_2\|pk_2\|ID_2$. If z_1, z_2 come from the same generator so that $ID_1 = ID_2 = ID$ (e.g., if an attacker substituted the components), then the problem rests with the signature s_1 hopefully being different from s_2. Recovering $x_1 = ID_1\|c_1\|r_1$ from s_1 and $x_2 = ID_2\|c_2\|r_2$ from s_2, we ultimately have a difference, as in case the generator is the same, the counters are different by construction. In case the generators are different, the two IDs are different too. It follows that the entire output of the generator, regardless of adversarial influence at any postprocessing stage – excluding the signature generation – is unique. We refer the interested reader to [7] for a comprehensive discussion.

b) Authenticity: Having stripped all layers of signatures and encryptions as sketched above, we are left with two identity strings ID and ID' when we reach the innermost piece of data $x = ID\|c\|r$, being wrapped inside $s\|pk\|ID'$. One indicator of an attacker having made changes is a mismatch between ID and ID'. However, a stronger indication is provided by the digital signature verification, which is the primary measure to assure authenticity. At this point, it is important to stress the need for the manufacturer's certificate that links the public key of the generator to its ID (for otherwise, an attacker could create his own signature key pair and trick the user of the random number generator into using the wrong key to check authenticity). The certificate can be standard (say, X.509), such as used in most conventional public-key infrastructures.

c) Entropy and Min-Entropy: Notice that besides randomness that possibly went into the signature (e.g., if a Nyberg-Rueppel signature was in charge) or later stages of the postprocessing (i.e., the key k), the assured entropy coming out of the quantum random generator is limited by what has been authenticated. Hence, only the innermost value r can be used to lower-bound the entropy of the final output (assuming possible adversarial modifications), leading to the entropy bound $H(z) \geq H(r)$.

Besides Shannon-entropy H, min-entropy H_∞ of the generator's output may be of interest, as most applications demand high min-entropy for matters of randomness extraction. This is most easily done by extracting the authenticated quantum random bitstring r from the generator's output z. By our construction, it is possible to recover the true randomness from the generator's output, thus the above inequality holds in exactly the same fashion for H_∞ in place of H. This can be proven easily, as all processing functions are injective by construction and thus cannot lower the min entropy. These considerations lead to the min-entropy bound $H_\infty(z) \geq H_\infty(r)$.

We stress that the injectivity of the signature is vital for this bound to hold, and the inequality could be violated if the signature with message recovery were replaced by a conventional signature (for a hash-then-sign paradigm, the lack of injectivity in the hash function would invalidate the above argument).

IV. Security and Efficiency

Roughly, the postprocessing stage adds some redundancy to the randomness r, which depends on the specific implementations of the signature and encryption. In case of RSA and AES, we end up with (currently [8]) 4096 bits for $R(ID\|c\|r)$. Defining $R(m) = m\|h(m)$, where h is a 256-bit cryptographic hash function like the SHA-2 (Secure Hash Algorithm 2), and using a 128 bit counter as well as an 80 bit ID (e.g., an ICCSN in a smartcard taking 10 bytes), we are left with a remainder of $4096-256-128-80 = 3632$ bits of raw quantum randomness r. Attaching the ID and a short RSA public key pk (16 bits), we expand the input via AES-CBC (cipherblock chaining (CBC) with ciphertext stealing) to $4096 + 80 + 16 = 4192$ bits. Concatenating another 128 bits for the AES key k yields final output of $4192 + 128 = 4320$ bits, among which 3632 bits are pure quantum randomness. The relative overhead is thus $\approx 19\%$.

In addition, the application of digital signatures naturally puts the chosen signature scheme in jeopardy of a known-message attack. Assuming that the generator is tamper-proof, chosen- or adaptive chosen message attacks (cf. [9]) are not of primary danger in this setting. Yet, we strongly advice to take hardware security precautions to protect the secret key against physical leakage and backward inference. Nevertheless, to avoid an attacker replacing the randomness source by another one (with low entropy), the signature scheme must be chosen with care.

In the presented form, authenticity, i.e., protection of known-message attacks, is solely based on computational intractability properties. If one wishes employ information-theoretic security, the digital signature with message recovery may be replaced by a conventional Message Authentication Code (MAC), based on universal hashing and continuous authentication, as it is the case for quantum key distribution (QKD) [10]. There, an initial secret r_0 shared between the peers of a communication link, is used to authentically exchange another secret r_1, which is then used to authenticate the establishment of a further secret r_2, and so on. (in the application of [10], r_1 would be a quantum cryptographically established secret key).

We can play the same trick here by putting an initial secret r_0 in charge of authenticating the first random values emitted by our quantum random generator. Instead of signature with message recovery we will use a "MAC with appendix", i.e., we use a function $MAC : \{0,1\}^* \times \mathsf{K} \to \{0,1\}^\ell$ (e.g., a universal hash-family [11]) to authenticate the string $ID\|c\|r_1$ by concatenating a keyed checksum as $R(ID\|1\|r_1)\|MAC(R(ID\|0\|r_1), r_0)$, when r_1 is the first random number ever emitted by our generator. After that, the authentication is done using the respective last number r_i, i.e., we emit $R(ID\|c+1\|r_{i+1})\|MAC(R(ID\|c\|r_{i+1}]), r_i)$, whenever r_{i+1} follows r_i in the sequence (see Figure 2 for an illustration).

However, we might run into issues of synchronization here, thus opening another potential attack scenario, when the adversary succeeds in blocking some of the random values. In that case, we would either have to attach multiple MACs and maintain a list of past authenticators, or periodically re-synchronize the process (which requires a fresh authentic key exchange with the generator). Hence, this variant may not necessarily be preferable in practical applications.

V. Conclusions

Applications that require high-quality random sources like quantum physics based ones, most likely do so because the upper level cryptographic application crucially rests on the statistical properties of the involved random quantities. Binding a random number to its origin is thus perhaps an overlooked precaution to avoid working with low-entropy or potentially coincidental random values in a cryptographic application. Interactive proofs of knowledge, as well as recent empirical findings [1] on parameter selection for RSA and the digital signature standard, dramatically illustrate the need for such post-processing.

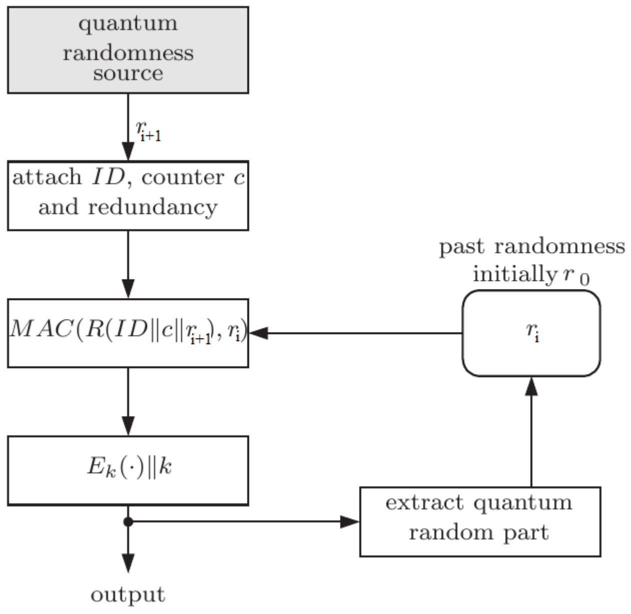
quantum randomness source
r_{i+1}
attach ID, counter c and redundancy
past randomness initially r_0
$MAC(R(ID\|c\|r_{i+1}), r_i)$
r_i
$E_k(\cdot)\|k$
extract quantum random part
output

Figure 2. Variant with continuous authentication instead of digital signatures

References

[1] A. K. Lenstra et al., “Ron was wrong, whit is right.”, Cryptology ePrint Archive, Report 2012/064, http://eprint.iacr.org/ [retrieved: July 7th, 2018].

[2] R. L. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems,” Commun. ACM, vol. 21, no. 2, 1978, pp. 120–126.

[3] T. ElGamal, “A public key cryptosystem and a signature scheme based on discrete logarithms,” in Proceedings of CRYPTO 84 on Advances in cryptology. New York, NY, USA: Springer New York, Inc., 1984, pp. 10–18.

[4] G. Locke and P. Gallagher, “Digital Signature Standard (DSS),” Federal Information Processing Standards (FIPS), Tech. Rep. FIPS PUB 186-3, 2009.

[5] A. Menezes, P. C. van Oorschot, and S. Vanstone, Handbook of applied Cryptography. CRC Press LLC, 1997.

[6] ISO/IEC, “ISO/IEC 7812-1:2006 Identification cards – Identification of issuers – Part 1: Numbering system,” http://www.iso.org, ISO/IEC, 2006, [retrieved: July 7th, 2018].

[7] P. Schartner, “Random but system-wide unique unlinkable parameters,” Journal of Information Security (JIS), vol. 3, no. 1, January 2012, pp. 1–10, ISSN Print: 2153-1234, ISSN Online: 2153-1242. [Online]. Available: https://www.scirp.org/Journal/PaperInformation.aspx?PaperID=16723

[8] D. Giry, “Bluecrypt – cryptographic key length recommendation,” http://www.keylength.com/, October 2011, [retrieved July 7th, 2018].

[9] S. Goldwasser, S. Micali, and R. L. Rivest, “A digital signature scheme secure against adaptive chosen-message attacks,” SIAM J. Comput., vol. 17, no. 2, Apr. 1988, pp. 281–308, [retrieved: July 7th, 2018]. [Online]. Available: http://dx.doi.org/10.1137/0217017

[10] G. Gilbert and M. Hamrick, “Practical quantum cryptography: A comprehensive analysis (part one),” 2000, uRL: http://arxiv.org/pdf/quant-ph/0009027 [retrieved: July 7th, 2018].

[11] M. Wegman and J. Carter, “New hash functions and their use in authentication and set equality,” Journal of Computer and System Sciences, 1981.