Nonce Generators and the Nonce Reset Problem

Erik Zenner

Department of Mathematics
Technical University of Denmark
e.zenner@mat.dtu.dk

Abstract. A nonce is a cryptographic input value which must never repeat within a given context. Nonces are important for the security of many cryptographic building blocks, such as stream ciphers, block cipher modes of operation, and message authentication codes. Nonetheless, the correct generation of nonces is rarely discussed in the cryptographic literature.
In this paper, we collect a number of nonce generators and describe their cryptographic properties. In particular, we derive upper bounds on the nonce collision probabilities of nonces that involve a random component, and lower bounds on the resulting nonce lengths.
We also discuss an important practical vulnerability of nonce-based systems, namely the nonce reset problem. While ensuring that nonces never repeat is trivial in theory, practical systems can suffer from accidental or even malicious resets which can wipe out the nonce generators current state. After describing this problem, we compare the resistance of the nonce generators described to nonce resets by again giving formal bounds on collision probabilities and nonce lengths.
The main purpose of this paper is to provide a help for system designers who have to choose a suitable nonce generator for their application. Thus, we conclude by giving recommendations indicating the most suitable nonce generators for certain applications.

Keywords: Cryptography, Security Engineering, Nonce, Nonce Reset, Nonce Generator

1 Introduction

Nonces are cryptographic inputs with the property that each value only occurs once within a given context[1]. Many modern cryptographic algorithms require a key and a nonce as input, and as long as the key is unchanged, the nonce must not repeat. Examples for cryptographic solutions that require nonces are stream ciphers, certain block cipher modes of operation, some message authentication codes (in particular Wegman-Carter based codes [2]), and certain entity authentication solutions.

[1] The term "nonce" is sometimes understood by cryptographers to be an abbreviation for "number used once". Even though this etymologically incorrect [1], it is a useful mnemonic for cryptographic purposes.

One possibility of generating nonces is the use of a random number generator (RNG). However, in order to avoid collisions, the nonce length has to be large, which may be problematic particularly in light-weight cryptographic systems with limited memory or bandwidth. In addition, a cryptographically strong RNG is not always available. Thus, a popular solution is to use a deterministic, stateful generator that keeps track of the nonces already used. The most obvious candidate for such a generator is a simple counter. As long as the generator does not "wrap around" (i.e. reaches a value that is longer than the nonce length, forcing it to start from 0 again), such a generator is good enough for most practical purposes.

However, this is only the case as long as the generator actually maintains its inner state. While this seems trivial in theory, it can not be taken for granted in practice. An unexpected power-down can mean the loss of all information that was not stored in non-volatile memory, and for many applications, constantly storing the nonce to Flash memory or a hard disk is not an option. For such systems, solutions are required that guarantee the nonce property also after a system reset.

Prior Art: Even though nonces play a prominent role in cryptography, hardly any literature exists on the issue. An overview of some known nonce techniques can be found in a discussion threat in the CFRG mailing list from early 2007 [3]. The use of nonces in security proofs was modeled by Rowaway [4]. In addition, a number of practical cryptosystems have been broken due to errors in the nonce handling [5–8].

Contribution: While the nonce generators described in this paper have been used in practice, our main contribution is the derivation of concrete bounds for collision probability and the nonce length. To the best of our knowledge, this is the first time that a full formal treatment of popular nonce generator techniques is given. We also give the first scientific discussion of the nonce reset problem and analysis of the techniques used to address it. Using the results of this paper, system designers can compare the suitability of the different nonce generators for their target application and make choices based on mathematical bounds.

Paper Structure: In Section 2, we review a number of nonce generators, most of which are well-known in the literature. We derive formal upper bounds on the nonce collison probabilities and lower bounds on the nonce lengths. Then we proceed to describe the nonce reset problem in Section 3. Here we also introduce a number of solutions to the problem, again giving formal bounds on collision probability and nonce length. Finally, in Section 4, we compare the nonce generators proposed and conclude the paper.

Notation: Throughout this paper, we will use the following variables. The maximum number of nonces produced by a generator is denoted by θ. The maximum number of nonce resets is denoted by $r - 1$, i.e. r is the maximum number of (re-)initialisations. The collision probability is denoted by p_c, and the maximum

allowable collision probability is denoted by $p_{\max}$. Finally, the nonce length is denoted by l. If a nonce consists of a counter and a random part, then the lengths of these parts are denoted by l_1 and l_2, respectively.

In the algorithmic descriptions of Section 3, $a \leftarrow b$ denotes the assignment of value b to variable a, while $a = b$ denotes the logical comparison between a and b.

2 Standard Nonce Generators

Before discussing the nonce reset problem, we briefly review three basic types of nonce generators (NGs) and give bounds for the corresponding collision probabilities and nonce lengths.

Choosing the Right Nonce Length: It is important upon designing a nonce-based system to pick the right nonce length l. An *upper bound* for l often results from application limititations such as expensive bandwidth or storage. While it may be possible to choose any desired nonce length on e.g. a desktop or laptop computer, limitations may exist for resource-restricted devices. In light-weight systems such as smart cards, sensor nodes, RFID chips etc., non-volatile memory as well as transmission bandwidth is limited and expensive. Thus, a solution that simply chooses a large nonce to elimiate all potential problems is not an option in such a scenario – the nonce length has to be optimised as far as possible.

To this end, a *lower bound* for l is required. As it turns out, this lower bound depends on the type of NG used, as well as the maximum number θ of nonces required within one context. In Section 3, we will see that additional factors play a role if we also want the NG to address the reset problem.

Deterministic vs. Probabilistic NGs: Nonce generators can be either deterministic or probabilistic.

- Deterministic NGs use some kind of inner state to keep track of the values already used as nonces, ensuring that the same value never gets used again. Such generators have two functions: `Init()` is executed upon setting up the NG, while `Next()` outputs the next nonce value and updates the inner state.
- Probabilistic NGs use some kind of external randomness source to generate nonces. While all of them have a `Next()` function, some of them are stateless and do thus not require an `Init()` function. We denote them as "probabilistic" NGs because the sequences produced by them can in theory contain collisions. In practice, however, the collison probability p_c can be kept arbitrarily small by making the nonce length l large. Note that probabilistic NGs require a good RNG to function properly. Implementing a good RNG, however, is one of the hardest tasks in practical cryptography (see, e.g., [9]), and if the RNG is faulty, the true collision probability may be much higher than expected.

Additional Constraints: In some protocols in the literature, the NG is expected to have additional properties, such as unpredictability or pseudo-randomness. However, in this paper, we follow the cryptographically more rigorous view presented by Rogaway [4], namely that the role of a NG should be limited to guaranteeing collision-freeness. If additional properties are required, they have to be made explicit ("The protocol requires an pseudo-random nonce") and should be provided by the appropriate cryptographic primitives (e.g. a pseudo-random function) in a separate step. Thus, no such additional constraints are considered here.

2.1 Counter-based Generator

The most widespread deterministic generator is a simple counter. The `Init()` function consists of setting the counter `cnt` to 0 or to a random value, and the `Next()` function outputs `cnt` and increases it by 1 (modulo 2^l)[2].

Note that this type of generator is well-known and well-understood; we only repeat some known facts for completeness sake.

Nonce length: As long as the number θ of nonces drawn is at most 2^l, the output of a counter-based generator is guaranteed to be collision-free. This yields the trivial condition on the nonce length that $l \geq \log_2(\theta)$.

2.2 RNG-based Generator

The most common probabilistic NG simply outputs an l-bit random number `rnd` every time a nonce is requested. This NG does not maintain an inner state and thus, does not need an `Init()` function. As with the counter-based generator, the RNG-based generator is well-known and well-understood in the literature. Note that if a pseudo-RNG is used instead (as is often the case in practice), it should be a cryptographically secure one as formalised e.g. in [9]. If this security advice is heeded, it should not be possible to distinguish between the pseudo-RNG and a real RNG. Thus, in the following, the following facts for a real RNG can be applied to a cryptographically sound pseudo-RNG just as well.

Nonce length: The birthday bound (see e.g. [12, Section 6.6]) states that if θ out of 2^l elements are drawn in a mutually independent way, the collision probability p_c is upper bounded by $\frac{\theta^2-\theta}{2 \cdot 2^l}$.

[2] Another wide-spread type of deterministic NG is the use of the system clock [10, 11]. If there is at least one clock tick between two accesses to the `Next()` function, and if the clock is never reset or wrapped around, then this can be seen as a special case of a counter, where not every available nonce is actually used. However, there are additional problems, such as synchronisation problems or the possibility that someone (even inadvertedly) resets the system clock, thus creating a nonce re-use that goes unnoticed by the application.

This formula can be used to calculate the minimum length of a nonce. If $p_{\max}$ denotes the highest acceptable collision probability, we have:

$$\frac{\theta^2 - \theta}{2 \cdot 2^l} \leq p_{\max} \quad \Leftrightarrow \quad 2^l \geq \frac{\theta^2 - \theta}{2 \cdot p_{\max}}$$

Example: If we need at most $\theta = 2^{20}$ nonces and a collision probability of at most $p_{\max} = 2^{-20}$, then we get $2^l \geq 2^{59}$, meaning that the nonce has to have a minimum length of 59 bit. Thus, compared to the counter solution, the nonce has to be almost three times as long[3].

2.3 Mixed Solution

Another possibility is to combine the two approaches above by concatenating an l_1-bit counter `cnt` and an l_2-bit random number `rnd` into one nonce of length $l = l_1 + l_2$. For every call to the `Next()` function, `cnt` will be increased by one, and a new random number `rnd` will be generated. Obviously, this has the disadvantages of the RNG-based solution, namely that an RNG is required and that there is a risk for collisions. However, the collision probability and thus the nonce length is reduced by the counter part, and the solution offers some advantages in the case of nonce resets (see Section 3).

While the mixed solution is used in practice, we are not aware of a thorough discussion in the cryptographic literature. Thus, we give a more detailed analysis of its properties in the rest of this section.

Nonce length: As for the RNG-based generator, the nonce length depends on the collision probability. Thus, we start by giving a general collision bound for the mixed solution.

Lemma 1. *Assume that the number 2^{l_1} of possible counter values divides the maximum number θ of required nonces. Then the collision probability for the mixed solution is 0 if $0 \leq \theta \leq 2^{l_1}$, and*

$$p_c \leq \frac{\theta^2 - \theta \cdot 2^{l_1}}{2 \cdot 2^l}$$

otherwise.

Proof. Let us simplify notation by writing $S = 2^l$, $S_1 = 2^{l_1}$, and $S_2 = 2^{l_2}$. Note that for $\theta \leq 2^{l_1}$, no collision can occur, since we are guaranteed to use a new counter `cnt` each time. Beyond that point, a collision can occur if for two nonces with the same counter `cnt`, the random part `rnd` also collides. If a total of θ nonces is output by this NG, then for each value of `cnt`, we have $\frac{\theta}{S_1}$ calls to

[3] In many cryptographic texts, the simplified rule $2^l \approx \theta^2$ is used, meaning that nonces are chosen to be *exactly* twice as long as in the counter case. This, however, ignores the influence of the acceptable collision probability $p_{\max}$.

the RNG, generating an l_2-bit random part. Thus, for each counter, the collision probability p'_c is bounded by the birthday bound as

$$p'_c \leq \frac{\left(\frac{\theta}{S_1}\right)^2 - \left(\frac{\theta}{S_1}\right)}{2 \cdot S_2} = \frac{\theta^2 - \theta S_1}{2 \cdot S_1^2 \cdot S_2}.$$

In total, a collision occurs if there is a collision for any of the S_1 counters, i.e. the total collision probability is bounded by

$$p_c \leq S_1 \cdot p'_c = S_1 \cdot \left(\frac{\theta^2 - \theta S_1}{2 \cdot S_1^2 \cdot S_2}\right) = \frac{\theta^2 - \theta S_1}{2 \cdot S_1 \cdot S_2} = \frac{\theta^2 - \theta S_1}{2S}.$$

Resubstituting S_1 and S, we obtain the desired bound. □

Corollary 1. *Assume that the number 2^{l_1} of possible counter values divides the maximum number θ of required nonces. If the maximum acceptable collision probability is $p_{\max}$, then the nonce length for the mixed solution has to be at least*

$$l \geq \log_2 \left(\frac{\theta^2 - \theta \cdot 2^{l_1}}{2 \cdot p_{\max}}\right).$$

A Cautionary Note: Note that the above estimate is only correct if S_1 divides θ. In situations where this is not the case and where θ is small compared to S_1, the bound on p_c may be too low. Figure 1 illustrates this problem for three sample setups, namely for $l_1 = 6, 10, 11$ (from left to right) and a total nonce length of $l = 20$. The correct value for p_c is shown with a solid line, while the above bound is shown with a dashed line[4]. As can be seen, the error gets larger with increasing l_1. In fact, the error is bounded by $\frac{S_1}{8S_2}$, i.e. it is rather insignificant for small values of l_1 but must not be ignored for large l_1. This result is proven in Appendix A.

A simple way of solving this problem when designing a system is to choose S_1 and θ such that S_1 divides θ. If S_1 is small compared to θ, this should not be a problem. If, on the other hand, S_1 is close to θ, it is probably worth increasing l_1 by a few bits such that $S_1 \geq \theta$, thus achieving a collision probability of 0.

3 The Nonce Reset Problem

In actual implementations, the inner state of a deterministic NG has to be stored between two calls to the `Next()` function. Basically, there are two possibilities:

- **Volatile memory (VM):** This type of memory requires power to maintain its state. Examples are various types of RAM, but also CPU registers. The problem with using this kind of memory is that the NG state will be lost when the system suffers a (planned or accidential) power-down.

[4] Note that the "break" in the curve for $l_1 = 10$ is not a plotting error, but a property of the probability function, which is always convex with the exception of the points where S_1 divides θ.

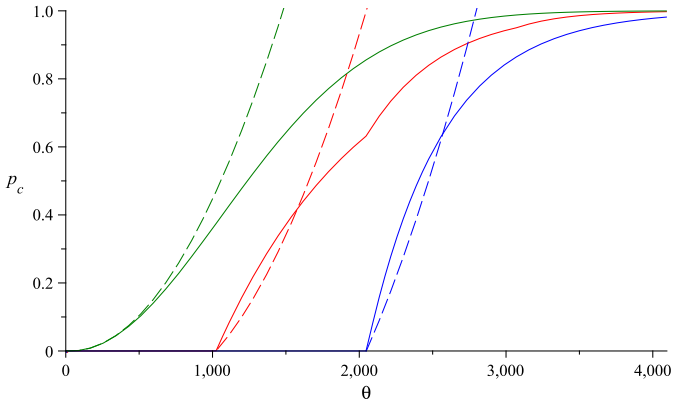


Fig. 1. Mixed Solution: Collision probabilities for $l = 20$ and $l_1 \in \{6, 10, 11\}$.

– **Non-volatile memory (NVM):** This type of memory maintains its state even if not powered. There are two types of solutions:
  - **Electronically addressed:** This includes technologies like EEPROM or Flash. They are rather expensive and slow compared to VM. As a result, on most platforms, designers will try to use as little electronically addressed NVM as possible.
  - **Mechanically addressed:** This includes typical "secondary" storage like magnetical or optical storage media (e.g. hard disks or DVDs). They have the disadvantage of being very slow compared to VM.

Long-term cryptographic keys are typically stored in NVM, and while they are in use, they are also loaded into VM. This way, they can be accessed fast and will nonetheless survive a system crash. For NGs, however, this solution is not always feasible. Electronically addressed NVM is often not available, and mechanically addressed NVM would slow down the system performance considerably due to the frequent changes of the NG state. Thus, practical solutions often store the NG state in volatile memory only. If, however, the key survives a system crash while the NG state does not, then some way of re-setting the NG is required.

It turns out that this reset function is often forgotten by NG designers. The classical mistake is to re-use the old key, but to start a new instance of the NG [5]. This means that the `Init()` function is called for the second time, which leads to a nonce re-use for deterministic NGs.

If the solution is built such that the cryptographic key survives a system crash, then the NG should have a `Reset()` function, which may or may not be identical to the `Init()` function. If `Reset()` and `Init()` are different, then it is important to always remember the following master rule for nonce initialisation upon system start-up: *If no key exists, run* `Init()`*. If a key exists, run* `Reset()`*.*

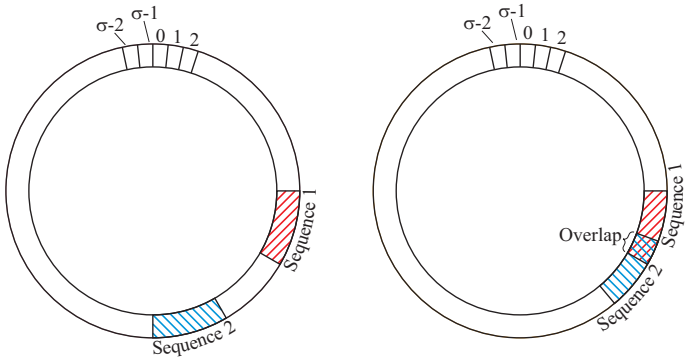


Fig. 2. The cycle of counter values

In the following, we will discuss a number of proposals for how to add a `Reset()` function to a counter-based NG. In addition, note that the simple RNG-based NG also solves the reset problem, albeit not in an optimal way.

3.1 Randomised Reset

A wide-spread solution is to reset the counter to a random l-bit value. Note that this solution requires an RNG, which has the disadvantages already discussed.

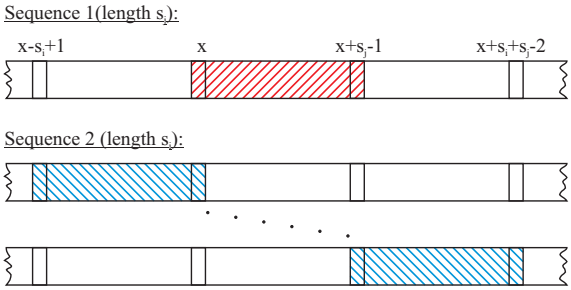
In addition, the solution is no longer deterministic and opens up for the possibility of nonce collisions. Note that since the `Next()` function computes the new counter as $i \leftarrow i + 1 \bmod 2^l$, counters will "wrap" if they get larger than $2^l - 1$. Thus, we can imagine the set of counters to be a cycle of length 2^l. Each sequence of counters between two resets marks a segment on this cycle, as illustrated in Figure 2. A collision between two sequences of counters occurs if those segments overlap, also shown in Figure 2.

Nonce length: A first intuition is that choosing the same nonce length as for an RNG-based NG would be save. But in this case, our new solution would not offer any advantages compared to an RNG-based NG. Thus, we are interested in showing that the required nonce length can be made smaller, as follows.

Lemma 2. *After $r - 1$ resets, the probability for at least one collision in a randomised reset solution is at most $\frac{r-1}{2^l}\left(\theta - \frac{r}{2}\right)$.*

Proof. We number the nonce sequences by $1, 2, \ldots, r$ and denote their respective lengths by $s_1, s_2, \ldots, s_r$. Before the first reset, there is only sequence 1, i.e. there can not be any collisions unless $s_1 \geq 2^l$.

Now consider the drawing of the starting point for sequence 2. Obviously, it must not collide with any of the s_1 points on sequence 1. In addition, it must not coincide with any of the $s_2 - 1$ points before sequence 1 either, since



If sequence 1 starts with nonce x, then sequence 2 will overlap if its starting nonce lies between $x - s_i + 1$ and $x + s_j - 1$ (both inclusive).

Fig. 3. Overlapping sequences

otherwise, the sequences will overlap (see Figure 3). Thus, a collision occurs with a probability of $\frac{1}{2^l}(s_1 + s_2 - 1)$.

For sequence 3, we already have to take the sequences 1 and 2 into account, and so on. In general, the probability p_i $(i \geq 2)$ for a new sequence to overlap with an already existing one is upper bounded as follows:

$$p_i \quad \leq \quad \frac{1}{2^l} \cdot \sum_{j=1}^{i-1} (s_j + s_i - 1).$$

The overall probability that at least one collision has occured after r sequences (i.e., $r - 1$ resets) is then upper bounded by

$$\begin{aligned} p_c \quad &\leq \quad \sum_{i=2}^{r} p_i \quad \leq \quad \frac{1}{2^l} \cdot \sum_{i=2}^{r} \sum_{j=1}^{i-1} (s_j + s_i - 1) \\ &= \quad \frac{1}{2^l} \cdot \left((r-1) \sum_{i=1}^{r} s_i - \sum_{i=1}^{r-1} i \right) \\ &= \quad \frac{1}{2^l} \cdot \left((r-1) \cdot \theta - \frac{r \cdot (r-1)}{2} \right) \\ &= \quad \frac{r-1}{2^l} \cdot \left(\theta - \frac{r}{2} \right) \end{aligned}$$

□

Note that this can be considered as a generalisation of the bound for purely random nonces. Purely random nonces correspond to random reset system with one reset after each output nonce, meaning $r = \theta$. In this case, the above collision bound becomes $\frac{\theta-1}{2^l} \cdot \left(\theta - \frac{\theta}{2}\right)$. This is the same as $\frac{\theta^2-\theta}{2 \cdot 2^l}$, which is the bound we already knew for purely random nonces.

Corollary 2. *Assume that a counter-based NG with randomised reset suffers at most $r-1$ resets during the lifetime of one key. If the maximum acceptable collision probability is $p_{\max}$, then we need*

$$l \geq \log_2\left(\frac{r-1}{p_{\max}} \cdot \left(\theta - \frac{r}{2}\right)\right).$$

Example: Consider the case of a resource-restricted device[5] with a built-in key and a maximum lifetime of 5 years ($2^{27.23}$ seconds). After each power-down, the device needs 30 seconds ($2^{4.91}$) to re-boot, which limits the number of possible resets to $r = 2^{22.32}$. On the other hand, if the system is running, it can send (due to bandwidth restrictions) at most 100 nonces ($2^{6.64}$) per second, i.e. up to $\theta = 2^{33.87}$ nonces in its lifetime. Thus, a naive application of the above corollary yields a minimum nonce length of $56.19 - \log_2(p_{\max})$ bit.

However, this approach overestimates the required nonce length. The reason is that the system can not be busy re-booting all the time while at the same time producing nonces all the time. In fact, the number $r-1$ of calls to the `Reset()` function and the number θ of calls to the `Next()` function depend on each other. An analysis of the function $f(r) = \log_2(\frac{r}{p_{\max}} \cdot (\theta - \frac{r}{2}))$ where θ is written as a function of r shows that the function is constantly increasing in the interval $[1, \theta]$. Accordingly, the maximum is reached for $r = \theta$. Since r has a known upper bound, we have $\theta = r = 2^{22.32}$, proving that a minimum nonce length of $43.64 - \log_2(p_{\max})$ bit is in fact sufficient.

3.2 Mixed Solution 1

The mixed solution described in Section 2.3 can also be used to solve the nonce reset problem. Again, an RNG is required, which may induce new problems into the solution.

Nonce Length: A general bound for the collision probability of this mixed solution can be given as follows.

Lemma 3. *After $r-1$ resets, the probability for at least one collision in the mixed solution from section 2.3 is bounded by*

$$p_c \leq \frac{\theta \cdot (\theta + 2^{l_1}(r-1))}{2 \cdot 2^l}.$$

Proof. We write again $S_1 = 2^{l_1}, S_2 = 2^{l_2}$ and $S = 2^l$. We start our analysis by observing that the worst case occurs if for each of $(r-1)$ resets, the same value is assigned to the counter part (as is the case in a counter solution without randomised reset). In this case, we have r sequences, each of which has a length of $\frac{\theta}{r}$ nonces. Thus, for no value of `cnt` we can have more than $\frac{\theta}{S_1 \cdot r} + 1$ assignments

[5] The example is taken from a real-world solution for intelligent homes.

to `rnd` between two resets, and the total number a of `rnd` values for each `cnt` is bounded by

$$a \;\leq\; r \cdot \left(\frac{\theta}{S_1 \cdot r} + 1\right) \;\leq\; \frac{\theta}{S_1} + r.$$

For each value of `cnt`, this means that the collision probability is upper bounded using the birthday bound by

$$p'_c[j] \;\leq\; \frac{a_j^2 - a_j}{2S_2}.$$

Consequently, the total collision probability for all S_1 values of `cnt` is bounded by

$$p_c \;\leq\; \sum_{j=1}^{S_1} \frac{a_j^2 - a_j}{2S_2} \;=\; \frac{1}{2S_2}\left(\sum_{j=1}^{S_1} a_j^2 - \sum_{j=1}^{S_1} a_j\right).$$

The sum $\sum_{j=1}^{S_1} a_j^2$ with a term sum of θ and terms in an interval $[0, \ldots, \frac{\theta}{S_1} + r]$ can be shown to be upper bounded by $\theta \cdot \left(\frac{\theta}{S_1} + r\right)$. In addition, it holds that $\sum_{j=1}^{S_1} a_j = \theta$. Thus, the bound can be computed to be

$$p_c \;\leq\; \frac{1}{2S_2}\left(\theta \cdot \left(\frac{\theta}{S_1} + r\right) + \theta\right) \;=\; \frac{\theta \cdot (\theta + S_1(r-1))}{2S}.$$

By resubstituting S_1 and S, we obtain the desired result. □

Note that this bound is a special case of the bound for the mixed solution without nonce reset, since for $r = 0$, we obtain $p_c \leq \frac{\theta \cdot (\theta - 2^{l_1})}{2 \cdot 2^l} = \frac{\theta^2 - \theta 2^{l_1}}{2 \cdot 2^l}$, which is exactly the bound for the mixed solution from Section 2.3.

Corollary 3. *If the maximum acceptable collision probability is $p_{\max}$, then the minimum nonce length for the mixed solution from section 2.3 is*

$$l \geq \log_2\left(\frac{\theta \cdot (\theta + 2^{l_1}(r-1))}{2 \cdot p_{\max}}\right).$$

3.3 Mixed Solution 2

An alternative is to modify the mixed solution described in Section 2.3 as follows: For every call to the `Next()` function, only the `cnt` part is updated. On the other hand, for every call to the `Init()` or `Reset()` function, the `cnt` part is set to 0, and the `rnd` part is set to a random value which is preserved until the next reset.

function `Init()`	**function** `Reset()`	**function** `Next()`
1. $i \leftarrow 0$	1. retrieve p from NVM	1. if $i = p$
2. $p \leftarrow u$	2. $i \leftarrow p$	2. $p \leftarrow p + u$
3. store p in NVM	3. $p \leftarrow p + u$	3. store p in NVM
	4. store p in NVM	4. output i
		5. $i \leftarrow i + 1$

Fig. 4. Counter-based NG using reset points

Nonce Length: This solution can be made very resistant against nonce resets by choosing the parameters as follows:

- If $2^{l_1} \geq \theta$, the construction is completely resistant against collisions as long as no nonce resets occur.
- If $2^{l_2} \geq \frac{r^2-r}{2 \cdot p_{\max}}$, the probability for a collision in case of a reset will be less than $p_{\max}$.

Thus, the recommended nonce length for this solution is $l \geq \log_2 \left(\theta \cdot \frac{r^2-r}{2 \cdot p_{\max}} \right)$ for $r > 1$. Note that if the maximum number $r - 1$ of expected resets is small compared to the total number of nonces, this solution is superior to mixed solution 1.

3.4 Reset Points

A completely different solution is to use reset points. This means that instead of using random start values after a reset, deterministic values are used in such a way that collision-freeness can be guaranteed. This is achieved by occasionally storing a safe reset point to non-volatile memory[6]. If this is done only rarely, the slow hardware access has little impact on the overall system performance.

To this end, we choose an interval size u which defines the distance between two reset points. If no reset occurs after u calls to the `Next()` function, a new reset point is stored. If, on the other hand, a reset occurs, the counter is set to the last stored reset point. Figure 4 describes this solution.

Nonce length: Note that the last nonce value ever to be produced by the system reaches its maximum if for each of the $r - 1$ calls to the function `Reset()`, a full u nonce values go unused. This means that even the largest nonce will be less than $\theta + (r-1) \cdot u$, and that the nonce length has to be at least $\log_2(\theta + (r-1) \cdot u)$.

Note that this is a generalisation of the nonce length given for simple counter-based NGs. If the system suffers no resets, then $r - 1 = 0$, and the above formula yields the well-known nonce length of $\log_2(\theta)$.

[6] This technique is mentioned in passing by Bernstein in [13], where he writes: "Store a safe nonce value – a new nonce larger than any nonce used – on disk alongside the key."

4 Comparison and Conclusions

4.1 Comparison

Table 1 compares the collision bounds for the solutions described above. All probabilities are given under the assumption that the actual number of nonces produced and actual number of nonce resets occuring do not exceed the anticipated values θ and $r-1$, respectively. The table also indicates whether an RNG is required. Remember that if this is the case, there is a probability (albeit low when choosing the right parameters) of producing a nonce collision. For these cases, the table indicates whether one collision significantly increases the risk of getting a whole sequence of collisions.

	coll. prob. without reset	coll. prob. with reset	RNG required?	colliding sequences
Counter w. rand. reset	$p_c = 0$	$p_c \leq \frac{r-1}{2^l}\left(\theta - \frac{r}{2}\right)$	yes	yes
RNG-based nonce	$p_c \leq \frac{\theta^2-\theta}{2\cdot 2^l}$	$p_c \leq \frac{\theta^2-\theta}{2\cdot 2^l}$	yes	no
Mixed solution 1	$p_c \leq \frac{\theta^2-\theta\cdot 2^{l_1}}{2\cdot 2^l}$	$p_c \leq \frac{\theta\cdot(\theta+2^{l_1}(r-1))}{2\cdot 2^l}$	yes	no
Mixed solution 2	$p_c = 0$	$p_c \leq \frac{r^2-r}{2\cdot 2^l}$	yes	yes
Couter w. reset points	$p_c = 0$	$p_c = 0$	no	n.a.

Table 1. Comparison of nonce generators

If no nonce resets are to be expected, the simple counter-based NG (not contained in the table) is the optimal strategy, yielding a minimum nonce length and a collision probability of zero.

If, however, nonce resets can happen, then the choice of the optimal NG and its parameters depends on the application situation. However, it seems that for many applications, the use of nonce reset points offers an optimal strategy. If storing a reset point at regular intervals is an option, this solution gives a guarantee for collision-freeness while having the shortest nonce length ($\log_2(\theta + (r-1)\cdot u)$ bit) of all solutions discussed. In addition, it does not require a random-number generator, thus removing an often vulnerable component from the solution.

Where regular storing of reset points is not an option, the mixed solution 2 will often give good results. Note that for most systems, a nonce reset is a rare event, and for small values of r, mixed solution 2 provides a low collision probability and a low nonce length.

4.2 Conclusions

In this paper, we have collected and described a number of nonce generators that are used in practice. For all of these generators, we have derived formal

bounds on the collision probabilities and nonce lengths. In addition, we have described the nonce reset problem, given a theoretical analysis of suitable nonce generators and discussed their resistance against nonce resets. To the best of our knowledge, this is the first time that a full formal treatment of popular nonce generator techniques is given. In particular, we hope to have given system designers a toolbox for choosing the right nonce generator and parameters for their target application.

Acknowledgements

The author wishes to thank G. Leander and L.R. Knudsen for inspiring discussions, D. Wagner for helpful comments on an early draft of this paper, and several anonymous reviewers for proposed improvements.

References

1. Wiktionary: Nonce. http://en.wiktionary.org/wiki/nonce (2009)
2. Wegmann, M., Carter, J.: New hash functions and their use in authentication and set equality. Journal of Computer and System Sciences **22** (1981) 265–279
3. List, C.M.: Consequences of nonce reuse. http://www1.ietf.org/mail-archive/web/cfrg/ (2007)
4. Rogaway, P.: Nonce-based symmetric encryption. In Roy, B., Meier, W., eds.: Proc. FSE 2004. Volume 3017 of LNCS., Springer (2004) 348–359
5. Borisov, N., Goldberg, I., Wagner, D.: Intercepting mobile communications: The insecurity of 802.11. In: Proc. 7th International Conference on Mobile Computing and Networking, ACM (2001) 180–189
6. Kohno, T.: Attacking and repairing the WinZip encryption scheme. In: Proc. 11th ACM Conference on Computer and Communications Security (CCS '04), ACM Press (2004) 72–81
7. Sabin, T.: Vulnerability in Windows NT's SYSKEY encryption. (BindView Security Advisory, December 16, 1999, available from http://marc.info/?l=bugtraq&m=94537756429898&w=2)
8. Wu, H.: The misuse of RC4 in Microsoft Word and Excel. http://eprint.iacr.org/2005/007 (2005)
9. Barak, B., Halevi, S.: A model and architecture for pseudo-random generation with applications to /dev/random. In: Proc. 12th ACM Conference on Computer and Communications Security (CCS '05), ACM Press (2005) 203–212
10. Gong, L.: A security risk of depending on synchronized clocks. ACM operating systems review **26** (1992) 49–53
11. Neuman, B., Stubblebine, S.: A note on the use of timestamps as nonces. Operating Systems Review **27** (1993) 10–14
12. Shoup, V.: A Computational Introduction to Number Theory and Algebra. Cambridge University Press (2005)
13. Bernstein, D.: The Poly1305-AES message-authentication code. In Gilbert, H., Handschuh, H., eds.: Proc. Fast Software Encryption '05. Volume 3557 of LNCS., Springer (2005) 32–49 also available from http://cr.yp.to/mac.html#papers.

A Detailed Analysis of the Mixed Solution

In Section 2.3, the collision probability for the mixed solution was upper bounded by $\frac{\theta^2-\theta\cdot S_1}{2S}$. However, this bound only holds if S_1 divides θ; otherwise, the bound is too low. In the following, we derive a universal bound.

Theorem 1. *The collision probability for the mixed solution in Section 2.3 is upper bounded by* $\frac{\theta^2-\theta\cdot S_1}{2S} + \frac{S_1}{8S_2}$.

Proof. Let us start by introducing the following notation. We write $\theta = q\cdot S_1 + r$, where q and r are the unique quotient and remainder, resp., when dividing θ by S_1.

The exact collision probability for the mixed solution can be modelled as follows. Imagine that there are $S_1 = 2^{l_1}$ containers and $S_2 = 2^{l_2}$ balls. With each call i to the NG, one ball is drawn at random (with replacement) and thrown into i-th container. This means that after θ iterations, all S_1 containers contain q balls, and r containers contain one additional ball. Thus, the total exact collision probability can be described by the following formula:

$$p_c = 1 - \left(\prod_{i=1}^{q-1}\left(1-\frac{i}{S_2}\right)\right)^{S_1}\cdot\left(1-\frac{q}{S_2}\right)^{r}.$$

By using the approximation that $1-\prod(1-p_i) \le \sum p_i$ for $0 < p_i \le 1$, we can upper bound this probability as follows:

$$p_c \;\le\; S_1\cdot\sum_{i=1}^{q-1}\frac{i}{S_2} + r\cdot\left(\frac{q}{S_2}\right) \;=\; S_1\cdot\frac{q\cdot(q-1)}{2\cdot S_2} + \frac{rq}{S_2}.$$

Substituting S_2 by S/S_1, we obtain:

$$p_c \;\le\; S_1^2\cdot\frac{q\cdot(q-1)}{2\cdot S} + \frac{rqS_1}{S} \;=\; \frac{S_1^2(q^2-q)+2rqS_1}{2S}.$$

This bound is a correct bound in the sense that it is always larger than the correct probability function. Now let us consider the estimate given in Section 2.3:

$$\begin{aligned}\frac{\theta^2-\theta\cdot S_1}{2S} &= \frac{(q\cdot S_1+r)^2-(q\cdot S_1+r)\cdot S_1}{2S}\\ &= \frac{S_1^2(q^2-q)+2rqS_1+(r^2-S_1r)}{2S}.\end{aligned}$$

As we can see, this bound differs from the above by the term $\frac{r^2-S_1r}{2S}$. Since $r \le S_1$ by definition of r, this term is always < 0 with the exception of $r = 0$, in which case both functions are identical. Figure 5 illustrates this by showing the correct probability (dotted), the simplified bound (solid), and the correct bound (dashed).

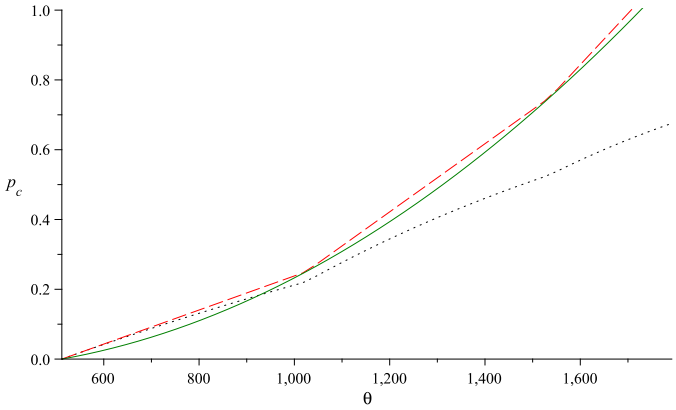


Fig. 5. Comparing correct and simplified bound for mixed solution ($l = 20, l_1 = 9$)

Analysis of the error function $\frac{r^2 - S_1 r}{2S}$ shows that it achieves its maximum for $r = \frac{S_1}{2}$, yielding a maximum error of $-\frac{S_1}{8S_2}$. By adding this maximum error to the simplified bound, we obtain a bound that is always correct and prove the theorem. □