Nonce Generators and the Nonce Reset Problem

Erik Zenner

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

Pisa, Sep. 9, 2009

Everyone knows...

Everyone knows what a nonce is:

- A nonce is a cryptographic value that is used only once.

Everyone knows...

Everyone knows what a nonce is:

- A nonce is a cryptographic value that is used only once.

Everyone knows what a nonce is used for:

- A nonce ensures that the cryptographic output for two identical key/message pairs looks different.

Everyone knows...

Everyone knows what a nonce is:

- A nonce is a cryptographic value that is used only once.

Everyone knows what a nonce is used for:

- A nonce ensures that the cryptographic output for two identical key/message pairs looks different.

Everyone knows how to generate a nonce:

- The simplest way to generate a nonce is to use a counter.

... can we go home now?

In theory...

In theory, the problem of nonces is solved.

Theory vs. practice:

- In theory, there is no difference between theory and practice.
- In practice, there is.

Outline

Outline

Strictly speaking...

Strictly speaking, a nonce does not exist.

- Is the number 213 a nonce?

Strictly speaking...

Strictly speaking, a nonce does not exist.

- Is the number 213 a nonce?

Being non-repeating is not a property of a number, but of a **sequence** of numbers or of the **algorithm** generating this sequence.

Nonce Generator (NG):

A **nonce generator** is a (deterministic or probabilistic) algorithm that outputs a sequence of numbers such that each number occurs at most once.

Note the similarities to random numbers!

What nonces aren't

The **only** property of the nonce is to be the output of a nonce generator.

- A nonce may be a public value.
- A nonce may be completely predictable.
- A nonce may have a lot of structure.

Formalisation (Rogaway, FSE 2004):

A **nonce-respecting adversary** is allowed to freely choose the nonces for his queries, as long as he does not choose the same nonce twice under the same key.

$\Rightarrow$ If you need anything stronger than that, don't call it a nonce!
$\Rightarrow$ It's also out of scope for this paper/talk.

Deterministic vs. probabilistic NGs

Deterministic nonce generator:

- The clean solution.
- All sequences output by this generator are nonce sequences.
- Classical example: Counter.

Probabilistic nonce generator:

- Behaves like a nonce generator most of the time.
- Some (few) sequences output by this generator contain repeating elements.
- Classical example: Random numbers.

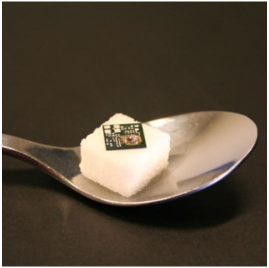
Outline

Motivational example

From a real-world consulting project:

- Low-cost sensor network system.
- Very little non-volatile memory available:
  - Enough to store the key.
  - Not enough to store the nonce.
- Frequent battery shut-down to save energy
  - $\Rightarrow$ Nonce state gets lost.
  - $\Rightarrow$ Counter-based system not feasible.
  - $\Rightarrow$ RNG-based nonces might save the day, but...
- Bandwidth is also very expensive:
  - $\Rightarrow$ Long nonces are prohibited.
  - $\Rightarrow$ RNG-based system not feasible.

How to solve this problem?



(c) Zensys A/S

The nonce reset problem

Nonces have to be stored somewhere:

	Volatile Memory	Non-volatile Memory
Examples	Registers, RAM	Harddisk, Flash
Speed	Fast	Slow
Available	Always	Sometimes
State loss?	Yes	No

Consequences:

- Nonces are generated and used in vol. memory
- Not always possible to store them in NV memory
- Vol. memory can lose state due to (voluntary or accidential) power-down
- Re-using same nonce after loss of nonce state can destroy cryptographic security!

Known solutions

Counter (deterministic):

- No randomness involved
- Keeping counter state is crucial
- If state is lost, the full nonce sequence is repeated

$\Rightarrow$ Risk of complete security break-down

Clock (deterministic):

- Special case of counter

Random nonces (probabilistic):

- RNG required
- Risk of collisions (birthday paradox)
- Larger nonce length ℓ required

$\Rightarrow$ Problematic if RNG not available or ℓ restricted

Other solutions?

Outline

Listing nonce generators

In the following:

- Give some sample nonce generators
- Not new, but knowledge badly documentet:
  - Google “random number generator” + cryptography: 124,000 hits
  - Google “nonce generator” + cryptography: 624 hits
    (mainly mailing lists and patent applications)
- List of nonce generators not exhaustive
- In the paper: Mathematics for choosing parameters

Counter with randomised reset (1)

Counter with randomised reset:

Minor modification of counter solution:

- Initialise to random value
- Upon reset, a new starting state is assumed

Advantages:

- No automatic repetition of nonce sequence upon reset

Counter with randomised reset (2)

Disadvantages:

- Requires an RNG
- If repetition happens: Partial sequence overlap

Mixed solution 1 (1)

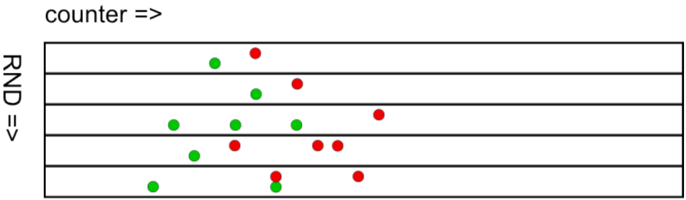
Mixed solution 1:

Known hybrid technique:

- Compose nonce of a counter **and** a random value
- Reset counter to random value

Advantages:

- Guaranteed no repetitions between two resets
- Collisions across two resets very unlikely
- No sequence overlap



Mixed solution 1 (2)

Disadvantages:

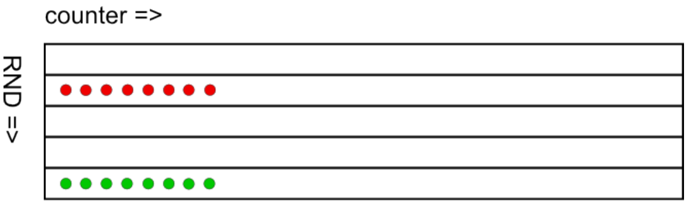
- Requires an RNG
- Nonce longer than pure counter, but shorter than random solution (for detailed mathematics: see the paper)

Mixed solution 2 (1)

Mixed solution 2:

Enhancement of mixed solution 1:

- Update the random value **only** upon reset.
- Set counter to 0 upon reset.



Mixed solution 2 (2)

Advantages:

- Collision probability for random part much smaller
- Random part can be kept small
  (again: see the paper for the maths)
- Total nonce size smaller than mixed solution 1

Disadvantages:

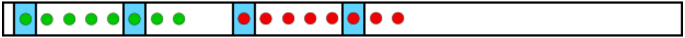
- Requires an RNG
- If RNG collision happens: Full sequence overlap

Reset points (1)

Counter with reset points:

If some NV memory is available:

- Use pure counter solution
- Store a larger counter value on NV memory
- Upon reset, continue from this larger counter



Reset points (2)

Advantages:

- With proper parameters: no collisions possible
- No RNG required

Disadvantages:

- Requires NV memory (can be smaller than nonce size)
- Nonce size slightly larger than for pure counter

Outline

How to compare?

In order to choose, be clear about your system requirements:

- Acceptable collision probability
- Acceptable nonce length
- Max. number of nonces required
- Max. number of system resets
- RNG available (how fast?)
- NV memory available (how fast?)
- Sequences overlap relevant?

All results in one table

	coll. prob. w/o reset	coll. prob. with reset	RNG ?	NVM ?	over-lap
CTR (standard)	0	1	no	no	full
CTR (rand. reset)	0	$\leq \frac{r-1}{2^l}\left(\theta - \frac{r}{2}\right)$	yes	no	part
CTR (reset pts.)	0	0	no	yes	-
RNG-based nonce	$\leq \frac{\theta^2-\theta}{2 \cdot 2^l}$	$\leq \frac{\theta^2-\theta}{2 \cdot 2^l}$	yes	no	no
Mixed solution 1	$\leq \frac{\theta^2-\theta \cdot 2^{l_1}}{2 \cdot 2^l}$	$\leq \frac{\theta \cdot (\theta+2^{l_1}(r-1))}{2 \cdot 2^l}$	yes	no	no
Mixed solution 2	0	$\leq \frac{r^2-r}{2 \cdot 2^l}$	yes	no	full

l = nonce length; l_1 = counter part length;
θ = max. number of nonces; r = max. number of (re-)inits

Conclusion

Best nonce generator depends on the circumstances:

- No nonce reset:
  - standard counter
- With nonce reset, NV memory available:
  - counter with reset points
- With nonce reset, RNG available:
  - random numbers if length does not matter
  - mixed solution 2 otherwise

Take side conditions (speed of RNG, speed of NV access, sequence overlap) into account.

Open problems

Some potential lines of work:

- List of nonce generators is not exhaustive.
- If neither RNG nor NV memory available:
  $\Rightarrow$ No solution to nonce reset problem available.
- Formal treatment of nonce generators in security proofs.
- Formal treatment of additional properties like unpredictability or pseudo-randomness.
- Formal separation of related terms like nonce, initialisation vector (IV), tweak, salt, pepper, challenge, freshness token, cryptosync,...

Thank you for your attention!