Standardization in Cryptography: the Case of Lightweight Cryptography

How do cryptographic primitives go from idea to deployment?

Léo Perrin

February 4, 2026

Contents

1 What Primitives are Deployed?

1.1 Where is cryptography deployed "in real life"?

Hard drive encryption `cryptsetup --help` gives:

> `LUKS: aes-xts-plain64, Key: 256 bits, LUKS header hashing: sha256, RNG: /dev/urandom`

Web Encryption on Firefox, head to a website whose URL starts with `https`, and then click on the lock. You will find something like Figure 1.

Navigo Pass modifications (e.g., new tickets bought) must be authenticated! See failures of the Dutch one[1]

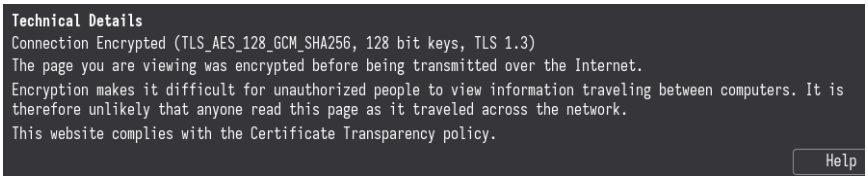


Figure 1: The TLS information for `https://hal.science`.

1.2 What is a primitive?

A cryptographic primitive is a very low level algorithm with a very simple API. Examples of symmetric primitives:

Block ciphers they map an element from $\mathbb{F}_q^n$ to an element from the same set. The mapping is parameterized by a key.

Hash functions map elements from $\mathbb{F}_q^*$ to $\mathbb{F}_q^d$,

There are many others: MACs, public permutations...

They are intended to provide **security** at low **cost**, especially for symmetric primitives which are much less costly than their public key counterparts.

Ultimately, they are parts of applications intended to provide:

Confidentiality an eavesdropper should not get any information from intercepted ciphertexts;

[1] `https://www.theregister.com/2008/04/16/dutch_transit_card_crippled/`.

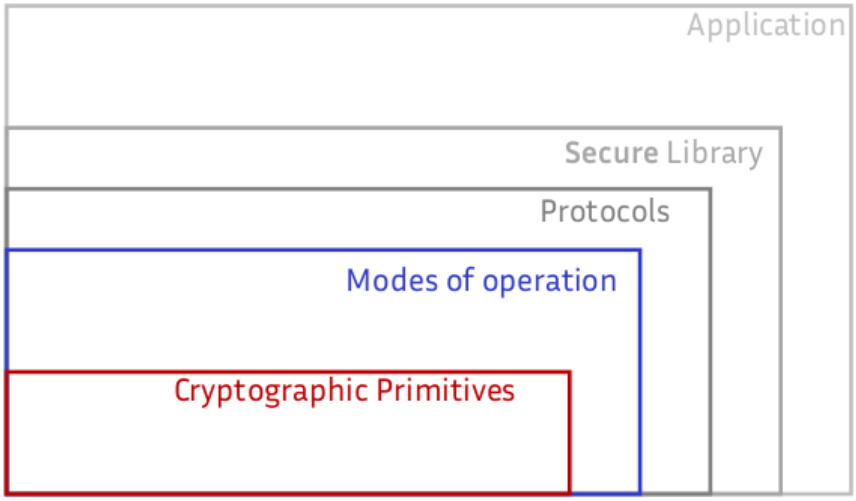


Figure 2: Where are the primitives?

Authentication the receiver should be sure that the message was indeed written by its author;

Integrity it should not be feasible to modify the message during transit in a way that is not noticed.

1.2.1 Examples of Modern Standard Primitives

Let us look at some common primitives.

AES Arguably one of the most common: the default choice for block ciphers. It was selected at the end of an open competition organized by the American NIST (National Institute for Standard and Technologies[2], which depends on the department of commerce).

SHA-x, x>1 The primitives with these names are hash functions, but are very different from one another. SHA-2 (in fact, SHA-256, SHA-384 and SHA-512) are built similarly to SHA-1 but are so far fine. SHA-3 is built completely differently (on purpose), and has a very wide security margin. Unlike the others, SHA-3 was selected after an open competition, again organized by NIST.

[2] https://www.nist.gov/.

Chacha20/Salsa20 Stream ciphers optimized for software implementation. The were designed withing the framework of the eStream project[3].

All of them are also part of the TLS standard, made by the IETF.

1.2.2 Examples of Proprietary/Legacy Primitives

Not all primitives used in practice are used because they have won a difficult competition. Others are the outcome of opaque processes where external input is not simply ignored, it is actively avoided. Examples include:

SHA-1 Designed by the literal NSA, now broken in practice [SBK+17]

Random Poprietary Algorithms car keys, OV-kart... See Table 3.

Arguably the most important such set of algorithms is intended for cell phones: A5/1, A5/2 for 2G, GEAx for satellite phones. These are standardized by ETSI[4]

2 The Life Cycle of a Primitive

Setting aside weird proprietary algorithms, the life cycle of a primitive is summarized in Figure 4.

The entities involved are:

- individual researchers, both from academia (i.e. people like me, Anne Canteaut, Maria Naya-Plasencia or Christina Boura) or from the industry;
- the academic community at large, which in particular organizes the conferences and provides legitimacy for partial results;
- companies that will handle the deployment (like Mozilla for Firefox); and
- standardizing bodies, which are supposed to essentially organize this process.

[3] See wikipedia: https://en.wikipedia.org/wiki/ESTREAM.
[4] European Telecommunications Standards Institute.

Table 3: A summary of some proprietary/legacy lightweight primitives. Block ciphers are marked with "†", MACs with "‡" and unmarked primitives are stream ciphers.

Name	Intended platform	Key	IS	IV	Att. time	Reference
A5/1	Cell phones	64	64	22	2^{24}	[And94]
A5/2		64	81	22	2^{16}	[BBK08]
CMEA †		64	16–48	–	2^{32}	[WSK97]
ORYX		96	96	–	2^{16}	[WSD+99]
A5-GMR-1	Satellite phones	64	82	19	$2^{38.1}$	[DHW+12]
A5-GMR-2		64	68	22	2^{28}	[DHW+12]
DSC	Cordless phones	64	80	35	2^{34}	[LST+09]
SecureMem.	Atmel chips	64	109	128	$2^{29.8}$	[GvRVWS10]
CryptoMem.		64	117	128	2^{50}	
Hitag2	Car key/ immobilizer	48	48	64	2^{35}	[VGB12]
Megamos		96	57	56	2^{48}	[VGE13]
Keeloq †		64	32	–	$2^{44.5}$	[BSK96]
DST40 †		40	40	–	2^{40}	[BGS+05]
iClass	Smart cards	64	40	–	2^{40}	[GdKGV14]
Crypto-1		48	48	96	2^{32}	[NESP08]
CSS	DVD players	40	42	–	2^{40}	[BD04]
Cryptomeria †		56	64	–	2^{48}	[BKLM09]
CSA-BC †	Digital televisions	64	64	–	2^{64}	[WW05]
CSA-SC		64	103	64	$2^{45.7}$	
PC-1	Amazon Kindle	128	152	–	2^{31}	[BLR13]
SecurID ‡	Secure token	64	64	–	2^{44}	[BLP04]
E0	Bluetooth devices	128	128	–	2^{27}	[ZXF13]

Figure 3: The characteristics of some proprietary primitives (see [BP17] for more details).

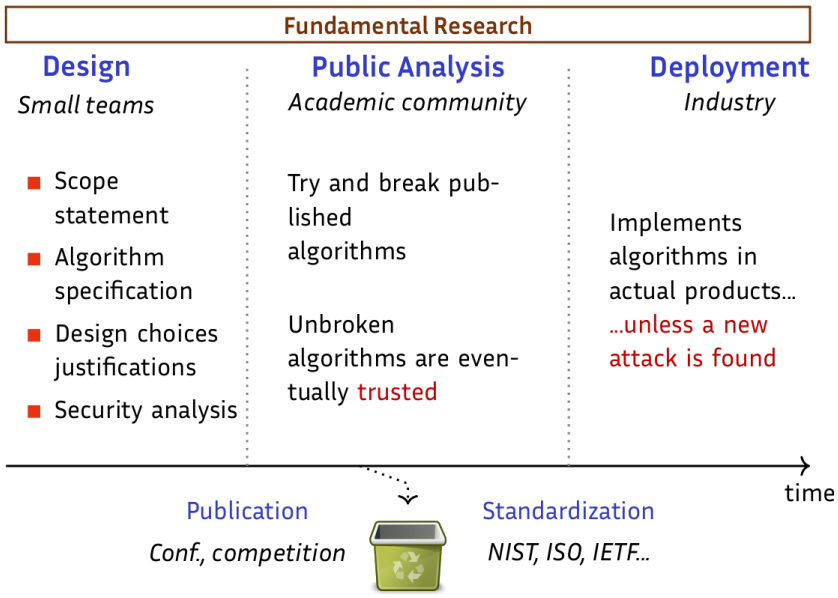


Figure 4: Life cycle of a primitive

2.1 Standardizing Bodies

As we have seen, setting aside the case of cell phones, a big difference between common (and good) algorithms and proprietary algorithms is their standardization by a reputable standardizing body.

These can be national or regional in nature (NIST, also one in Japan, China...), or international (ISO, IETF). The important thing to know is the **process** used to choose primitives: different bodies use different approaches.

NIST In practice, at this stage, the most important! Some of their standards are arguably the best (AES, SHA-3). Others, much less so (DUAL-EC), as we will see below. Decisions are ultimately made by NIST, after consulting with the NSA, however, public input is usually sought, and listened to.

ISO Essentially a diplomatic institution: often, one country/one vote. Decisions go through a lot of red tape, which is not necessarily a bad thing. Public input is not sought, but we (academic) can find ways in the discussions.

IETF The Internet Engineering Task Force is a public forum/mailing list[5] where discussion are made, and eventually where standards are discussed. These are called "RFCs".

2.2 When primitives don't follow this cycle

The elephant in the room is the involvement of various governmental agencies. We know for example thanks to Snowden that the NSA had a significant budget (as in, many millions of dollars) allocated to tampering with cryptography standardization. More concretely, at this stage, we know that the following algorithms are highly suspicious.

A5/2 this stream cipher was part of the 2G standard, and has an effective key size of 2^{40} in way that can only be intentional. It is backdoored.

GEA this stream cipher was part of satellite phone standards, and has an effective key length of 2^{40} (see [BDL+21]). It is backdoored.

DUAL-EC this PRNG was standardized by the NIST under pressure from the NSA. See [BLN15] for more details on DUAL-EC itself, and [CCG+16] for a detailed example of why the whole idea was a bad one. It implements a very crude backdoor.

Current Russian Standards Streebog (hash function) and Kuznyechik (block cipher) are the current standards in Russia, and have been for about 10 years. Its designers have repeatedly lied in public about the generation process of the their S-box [Per19, BPT19].

That an algorithm is standardized is great but it absolutely **not** sufficient to trust it!

2.3 How can a primitive deserve trust?

In my opinion, it is wrong to think that a primitive should be seen as fine until it is broken. It is **entirely** on the designers to make sure that they provide solid security argument, and to ease third party cryptanalysis. An algorithm that is hard to study should, by construction, be considered insecure.

[5] In fact, you can sign up/check it for yourselves: https://datatracker.ietf.org/rg/cfrg/about/.

3 A Case Study: the Lightweight Cryptography Competition

https://csrc.nist.gov/Projects/lightweight-cryptography/

In the early 2010's, NIST started to express interest in standardizing "lightweight" algorithms. By that point, many such algorithms had been published at academic conferences (see Figure 5).

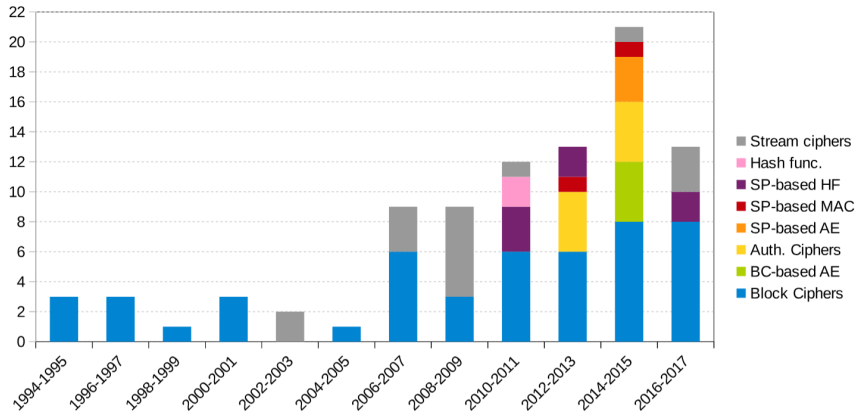


Figure 5: The number of "lightweight" symmetric primitives published at academic conferences between 1994 and 2017 (from [BP17]).

How did we go from these dozens of algorithms to a unique standard, Ascon[6]?

3.1 What is "Lightweight"?

First things first: we need to define "lightweight". It is much harder than it seems!

Here, "lightweight" should not be thought of as a property of the primitives but as one of the platforms intended to run them. It certainly shouldn't be a property of their security ("lightweight cryptography" is not "cryptographie faible", despite what some audacious official translations might lead you to believe).

[6] https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-232.pdf

3.1.1 The case of software

For software, a "lightweight" platform is a micro-controller[7], i.e., a small processor with limited frequency, limited word-size (as low as 8 bits), limited ROM and RAM, a limited amount of registers, and possibly a low amount of total energy available (small battery).

A primitive suitable for such environments must have the following properties.

Low code size the full implementation should not take too much space in the non-volatile memory since this space must be saved for the rest of the logic.

Low number of clock cycle/byte Since the frequency is low, a high number of clock cycles means a very slow encryption/decryption/authentication. It also means a higher amount of energy is used in total.

Low RAM consumption RAM consumption should a priori not be a big requirement (cryptographic primitives typically use a tiny amount), however, loading and fetching data in the RAM is much slower. Thus, RAM interactions must be kept at a minimum.

A primitive optimized for software must thus run well on 8-, 16- and 32-bit micro-controllers, and ideally fit in the registers of the smallest micro-controllers. A careful analysis of the number of cycles needed for each operation must also be made: on a laptop, rotating a 32-bit integer by any amount takes one CPU cycle. On micro-controllers, it depends: it can be literally free if the amount is a multiple of 8, or extremely expensive since some platforms only provide a shift by 1 for 8-bit words.

Which of the following lines is "lightweight"?

```
uint8_t lut[256] = {99, 124, 119, 123, ... 176, 84, 187, 22};
uint32_t state[2] = {0, 0};
for (unsigned int r=0; r<8; r++)
{
    state[0] ^= lut[state[1] & 0xFF];
    state[0] = ((state[0] << 12) | (state[0] >> 20)) & 0xFFFFFFFF;
    state[1] ^= state[0];
    state[0] += state[1] << 8;
}
```

[7] https://en.wikipedia.org/wiki/Microcontroller

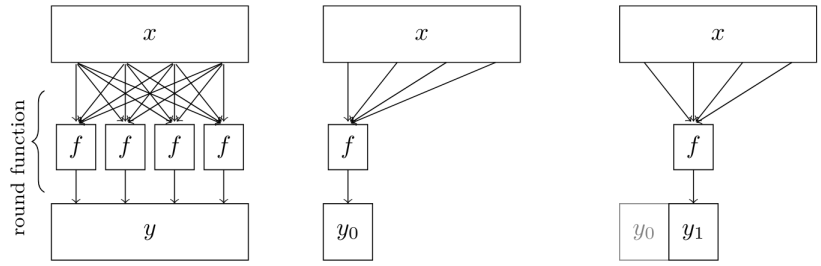
3.1.2 The case of hardware

This one is a bit more vague since "hardware" can mean different things. One of the main use cases corresponds to RFID tags. In general, in this case, we want to have the following properties.

Low total state size registers that can be updated are more expensive than fixed values, both in terms of circuit manufacturing cost and in terms of energy consumption.

Low number of logical gates similarly, logical gates (NAND, OR,...) are what costs.

On the other hand, complicated bit permutations are essentially free, and throughput is not really a concern. It is possible to further minimize the total number of gates using "serialization".



(a) Round-based ($t = 1$). (b) Serial ($t = 1$). (c) Serial ($t = 2$).

Figure 6: Principle of a serialized implementation.

Looking back at the previous lines, which of them are "lightweight", but for hardware?

```
uint8_t lut[256] = {99, 124, 119, 123, ... 176, 84, 187, 22};
uint32_t state[2] = {0, 0};
for (unsigned int r=0; r<8; r++)
{
    state[0] ^= lut[state[1] & 0xFF];
    state[0] = ((state[0] << 12) | (state[0] >> 20)) & 0xFFFFFFFF;
    state[1] ^= state[0];
    state[0] += state[1] << 8;
}
```

3.1.3 Side-Channel Attack Prevention

Side-channel attacks do not target an algorithm in some abstract sense, but rather a specific implementation. For instance, the electric consumption during the evaluation of `lut[x]` is correlated with the value of `x`. As a consequence, closely monitoring a circuit evaluating `lut[x ^ k]` for a secret key part `k` can reveal a lot of information about `k`.

To prevent such attacks, two directions must be considered at the same time.

Forbidden operations. Any operation that does not take a constant time, in particular table lookups, must be avoided at all cost!

Easing Counter-Measures Masking or threshold implementations are generic techniques that mitigate side-channel by randomizing parts of the computation. However, they are costly, and get costlier as the number of AND gates increases: this number should be minimized.

3.2 The Steps of the Competition

I wasn't just in the front row to watch this "competition" unfold, I was somewhere on the stage! How did it go? Let's see this through the history of SPARKLE [BBdS+20], a candidate I co-designed. Some of this The timeline can be also be seen on the NIST website for this process:

https://csrc.nist.gov/Projects/lightweight-cryptography

March 2014 During an informal presentation at the main symmetric cryptography conference (FSE), John Kelsey from NIST mentioned the start of a project on their side to standardize some "lightweight algorithms".

October 2016 NIST organized a first workshop on their campus[8]. I presented a block cipher I co-designed, SPARX [DPU+16]; the NSA presented SIMON and SPECK [BSS+13].

March 2017 NIST publishes a report[9] stating their intention to move forward with standardization.

April 2018 ISO refuses the standardization of SIMON and SPECK

[8] https://csrc.nist.gov/Events/2016/lightweight-cryptography-workshop-2016
[9] https://nvlpubs.nist.gov/nistpubs/ir/2017/NIST.IR.8114.pdf

July 2018 I was invited in Luxembourg by my former team and suggested we submitted a candidate based on improvements of SPARX.

Late 2018 NIST published a call for submissions[10] on a mailing list that had been setup a bit earlier[11]. However, the original timeline is messed up by an American government shutdown.

Late February 2019 56 round 1 candidates have been received, including SPARKLE,... and a SPECK-based candidate. Cryptanalysis starts.

August 2019 NIST already moves on to Round 2 by removing those clearly unfit for purpose. 32 are left.

June 2020 The academic community publishes a special issue of the ToSC journal[12] which gave an opportunity for Round 2 candidates to properly present their submission to the academic community.

March 2021 NIST announces 10 finalists. In my opinion, they made categories of primitives (do-it-all sponge, completely software optimized algorithm, completely hardware optimized algorithm, etc), and picked the most promising in each category. We asked two more implementers to join the SPARKLE team to help with hardware implementations.

Februry 2023 ASCON is announced as the winner of the competition.

August 2025 The standard describing ASCON is published.

Some remarks:

1. From its "official" start to its end, this process took **11 years**!
2. Of the 56 candidates, 55 will most likely never be used.
3. The cryptanalysis was significant at first, much slower in the end (the finalists were all solidly designed by teams of skilled cryptographers, algorithms with careless mistakes had been removed by that point).
4. ASCON is a **trusted** primitive.

[10] https://csrc.nist.gov/csrc/media/Projects/lightweight-cryptography/documents/final-lwc-submission-requirements-august2018.pdf

[11] https://csrc.nist.gov/projects/lightweight-cryptography/email-list

[12] https://tosc.iacr.org/index.php/ToSC/issue/view/182

5. Will deployment follow? We will see... The competition might have come too late, and the constraint niche of ASCON, while very large, is for the most part firmly occupied by the AES, which is even more trusted.

4 Conclusion

How do cryptographic primitives go from a mere idea to being deployed?

We can provide various points to answer this question:

1. An **overwhelming** majority of cryptographic primitives that are thought by cryptographers (however skilled they might be) are **never** deployed (and that's OK!).
2. The primitives that are deployed correspond to specific needs which can only be identified through discussion with intended "deployers", and which can only be satisfied through discussions with implementers.
3. It is necessary to have a deep understanding of the constraints imposed by intended use to build a primitive which is both "efficient" and "secure" since the meaning of both is highly context-dependent. For instance, what is "lightweight"?
4. Not all standards deserve trust, some in fact absolutely **do not** deserve any.

The exercises in `../exercises/` will allow you to see "in person" the impact the specifics of a cipher have on its suitability for implementation, and how a bad algorithm can be identified.

References

[BBdS+20] Christof Beierle, Alex Biryukov, Luan Cardoso dos Santos, Johann Großschädl, Léo Perrin, Aleksei Udovenko, Vesselin Velichkov, and Qingju Wang. Lightweight AEAD and hashing using the sparkle permutation family. *IACR Trans. Symmetric Cryptol.*, 2020(S1):208–261, 2020.

[BDL+21] Christof Beierle, Patrick Derbez, Gregor Leander, Gaëtan Leurent, Håvard Raddum, Yann Rotella, David Rupprecht, and Lukas Stennes. Cryptanalysis of the GPRS encryption algorithms GEA-1 and GEA-2. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part II*, volume 12697 of *Lecture Notes in Computer Science*, pages 155–183. Springer, 2021.

[BLN15] Daniel J. Bernstein, Tanja Lange, and Ruben Niederhagen. Dual EC: A standardized back door. Cryptology ePrint Archive, Paper 2015/767, 2015.

[BP17] Alex Biryukov and Léo Perrin. State of the art in lightweight symmetric cryptography. *IACR Cryptol. ePrint Arch.*, page 511, 2017.

[BPT19] Xavier Bonnetain, Léo Perrin, and Shizhu Tian. Anomalies and vector space search: Tools for s-box reverse-engineering. *IACR Cryptol. ePrint Arch.*, page 528, 2019.

[BSS+13] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK families of lightweight block ciphers. *IACR Cryptol. ePrint Arch.*, page 404, 2013.

[CCG+16] Stephen Checkoway, Shaanan Cohney, Christina Garman, Matthew Green, Nadia Heninger, Jacob Maskiewicz, Eric Rescorla, Hovav Shacham, and Ralf-Philipp Weinmann. A systematic analysis of the juniper dual EC incident. Cryptology ePrint Archive, Paper 2016/376, 2016.

[DPU+16] Daniel Dinu, Léo Perrin, Aleksei Udovenko, Vesselin Velichkov, Johann Großschädl, and Alex Biryukov. Design strategies for ARX with provable bounds: Sparx and LAX. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 484–513, 2016.

[Per19] Léo Perrin. Partitions in the s-box of streebog and kuznyechik. *IACR Trans. Symmetric Cryptol.*, 2019(1):302–329, 2019.

[SBK+17] Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov. The first collision for full SHA-1. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 570–596. Springer, 2017.