UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA

Lecture 9 - Lightweight Cryptography

Prof. Esposito Christian

::.. Summary

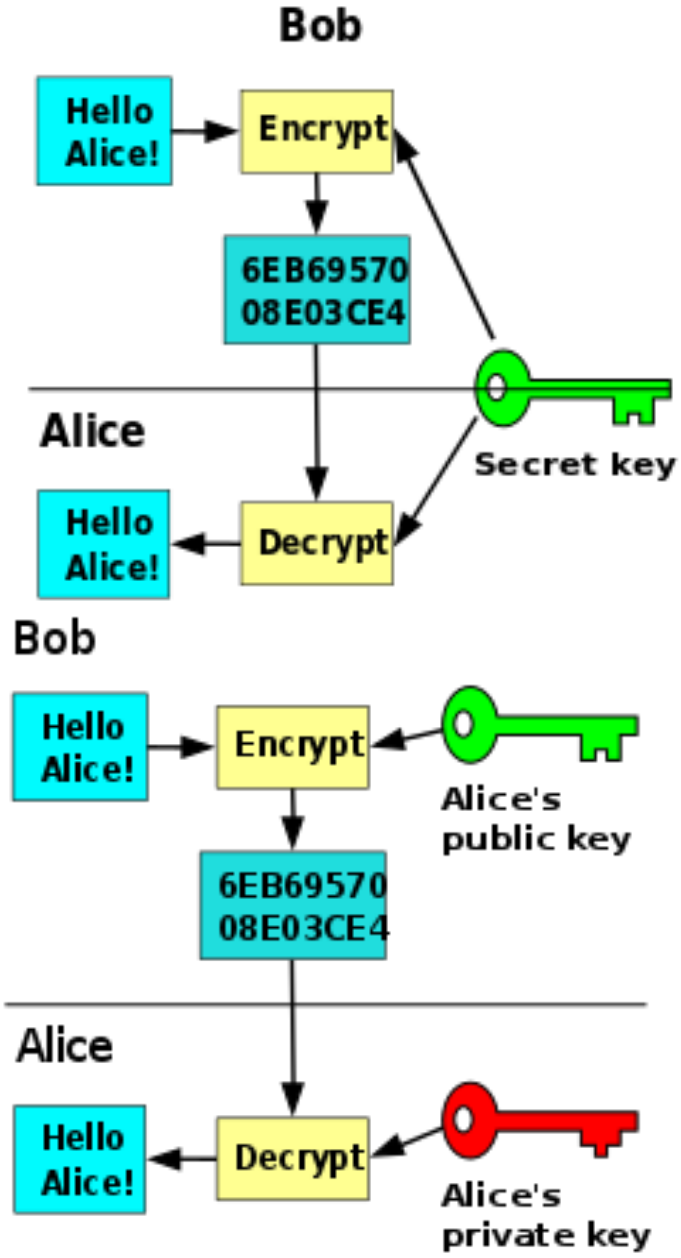
- Lightweight Cryptography
  - Introduction,
  - Solutions.
  - ECC

::.. Key Lectures

- Buchanan, William J., Shancang Li, and Rameez Asif. "Lightweight cryptography methods", Journal of Cyber Security Technology 1.3-4 (2017): 187-201.
- Dhanda, Sumit Singh, Brahmjit Singh, and Poonam Jindal. "Lightweight Cryptography: A Solution to Secure IoT", Wireless Personal Communications (2020): 1-34.

Introduction to Cryptography

::.. Cryptography (1/5)

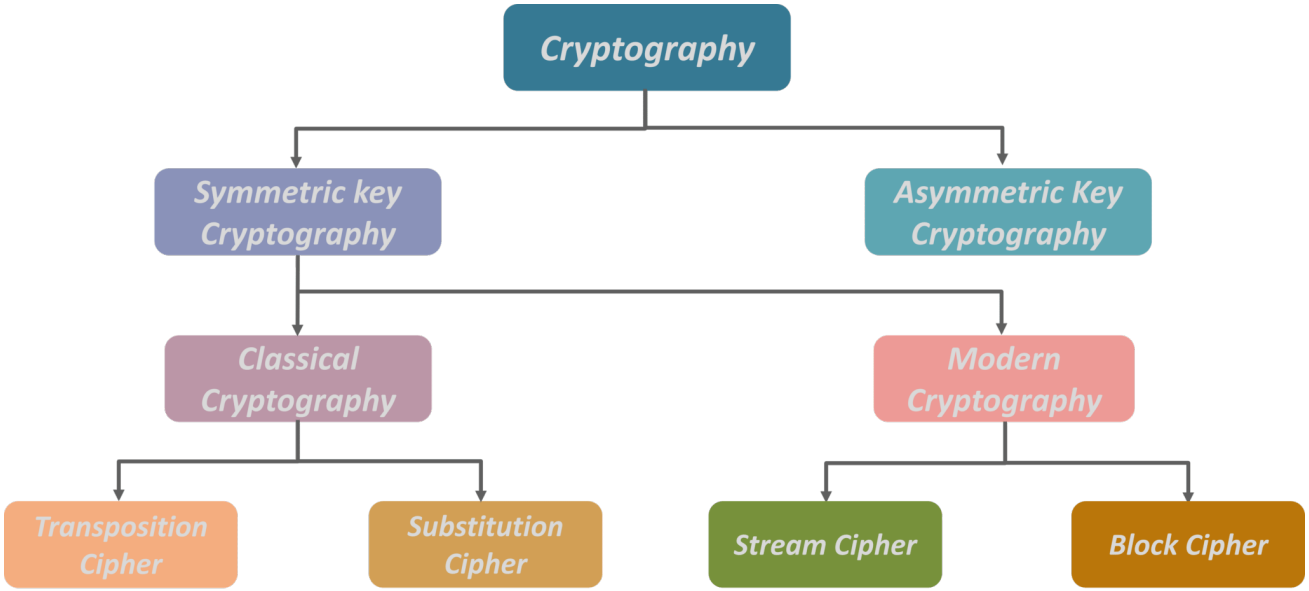


Cryptography is about constructing and analyzing protocols that prevent from reading private messages and guarantee various aspects in information security such as data confidentiality, data integrity, authentication, and non-repudiation.

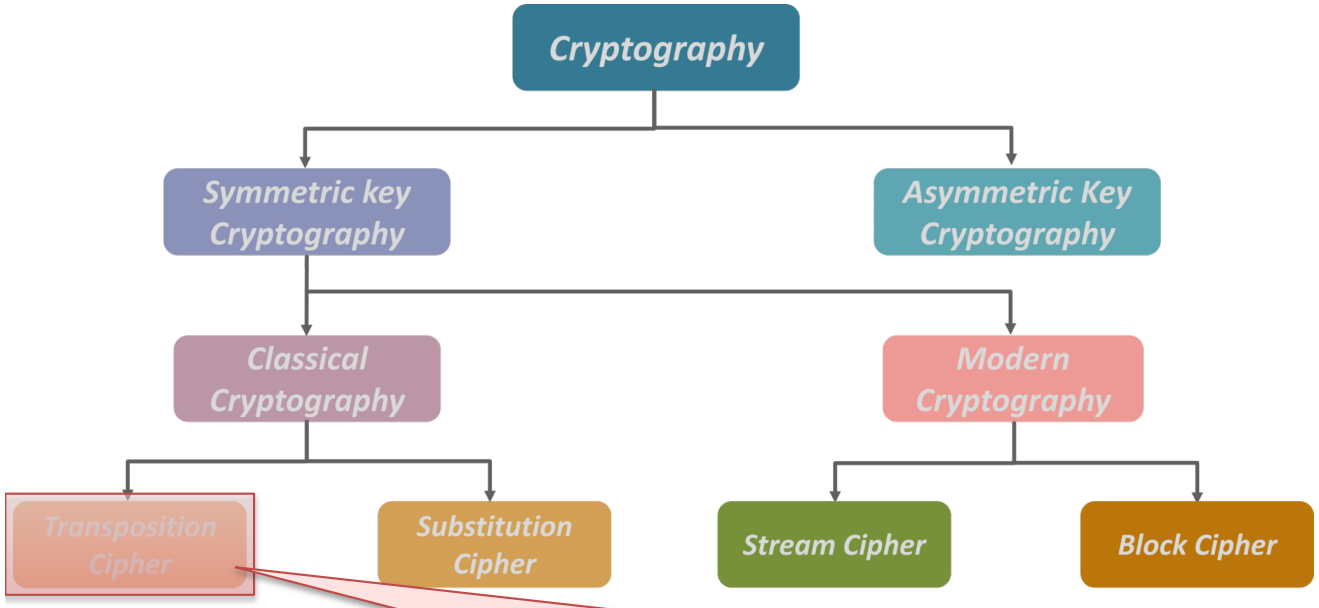
Encryption is the conversion of information from a readable state to apparent nonsense by using a mathematical function and a random string of bits being difficult to guess.

It is broadly classified into Symmetric key and Asymmetric key Cryptography.

::.. Cryptography (2/5)



::.. Cryptography (2/5)



The ciphertext constitutes a permutation of the plaintext.

1	2	3	4	5	6
M	E	E	T	M	E
A	F	T	E	R	P
A	R	T	Y		

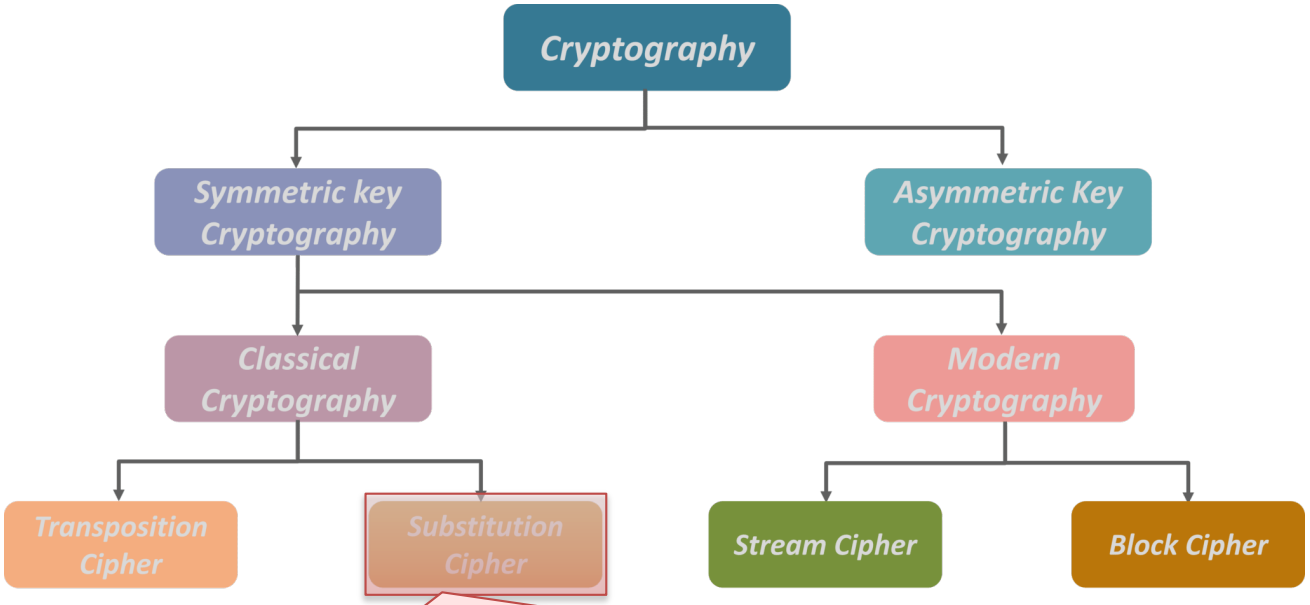
4	2	1	6	3	5
T	E	M	E	E	M
E	F	A	P	T	R
Y	R	A		T	

Plain Text: MEET ME AFTER PARTY

Key Used: 421635

Cipher Text: TEMEEMEFAPTRYRAT

::.. Cryptography (2/5)

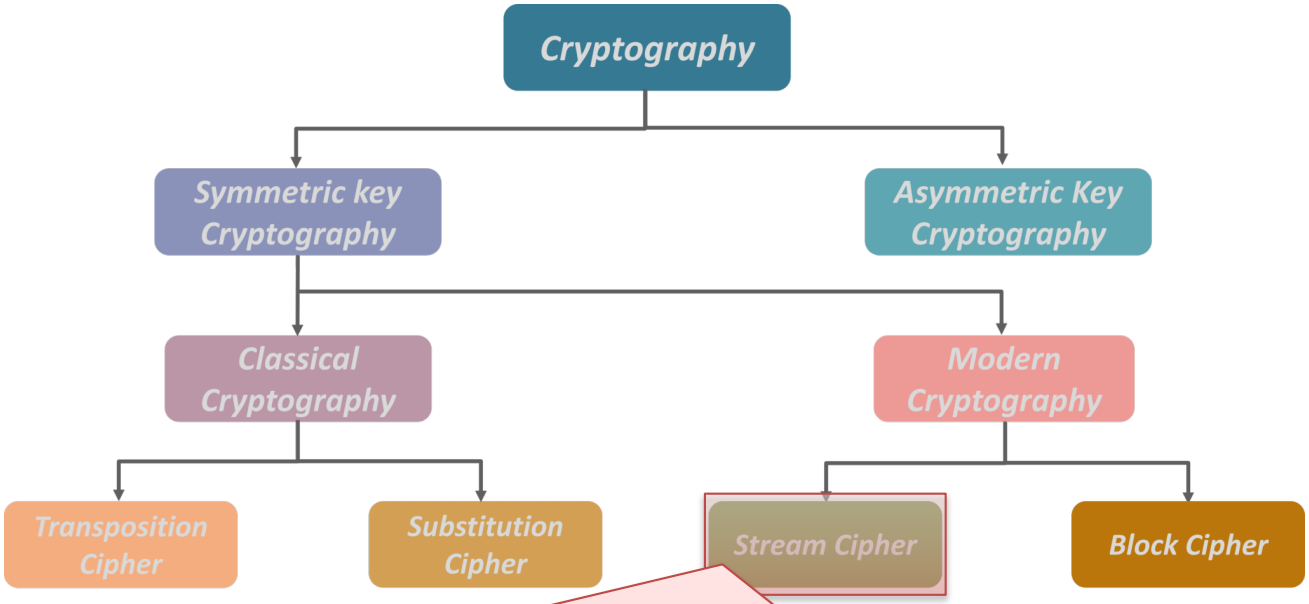


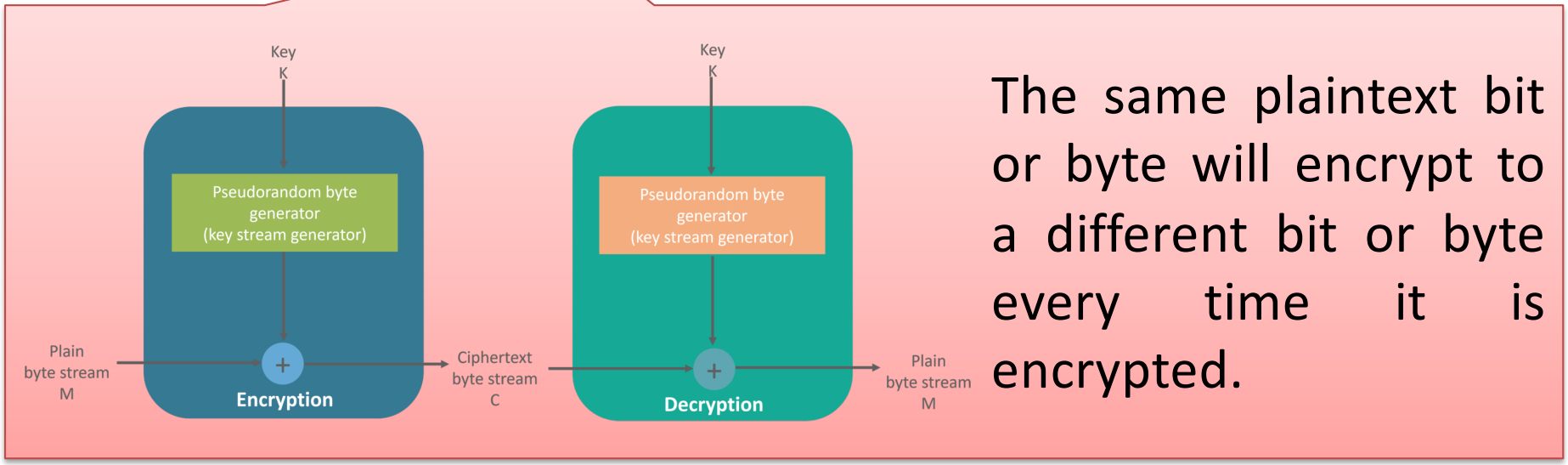
Units of plaintext are replaced with ciphertext, according to a fixed system.

Plaintext Alphabet:	ABCDEFGHIJKLMNOPQRSTUVWXYZ
Keyword:	Zebras
Ciphertext Alphabet:	ZEBRASCDFGHIJKLMNOPQTUVWXY

A message of: flee at once. We are discovered!
enciphers to: SIAA ZQ LKBA. VA ZOA RFPBLUAOAR!
SIAAZ QLKBA VAZOA RFPBL UAOAR

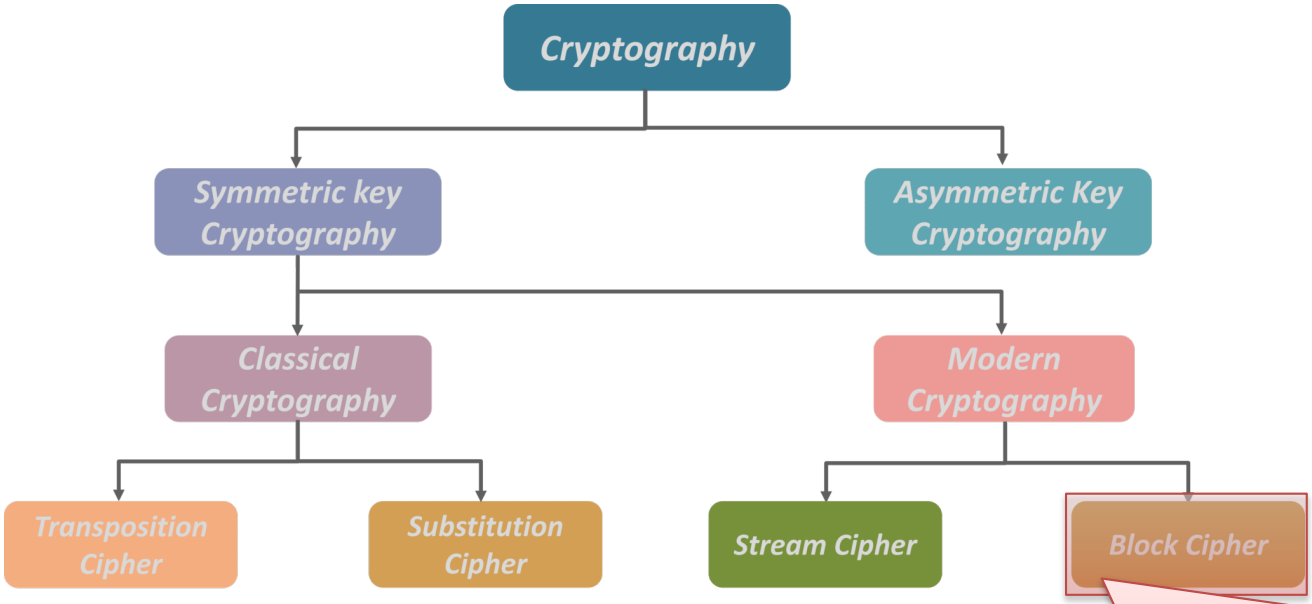
::.. Cryptography (2/5)

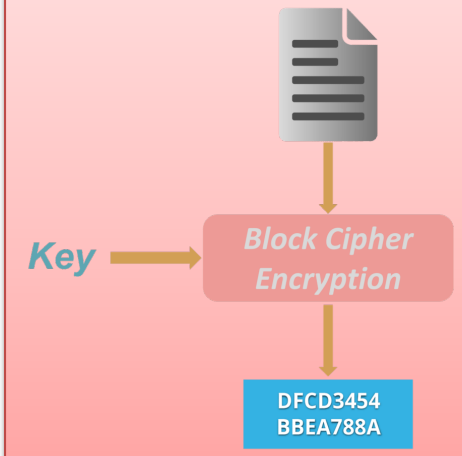


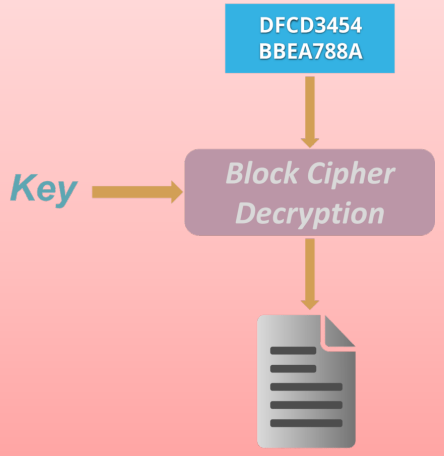


The same plaintext bit or byte will encrypt to a different bit or byte every time it is encrypted.

::.. Cryptography (2/5)

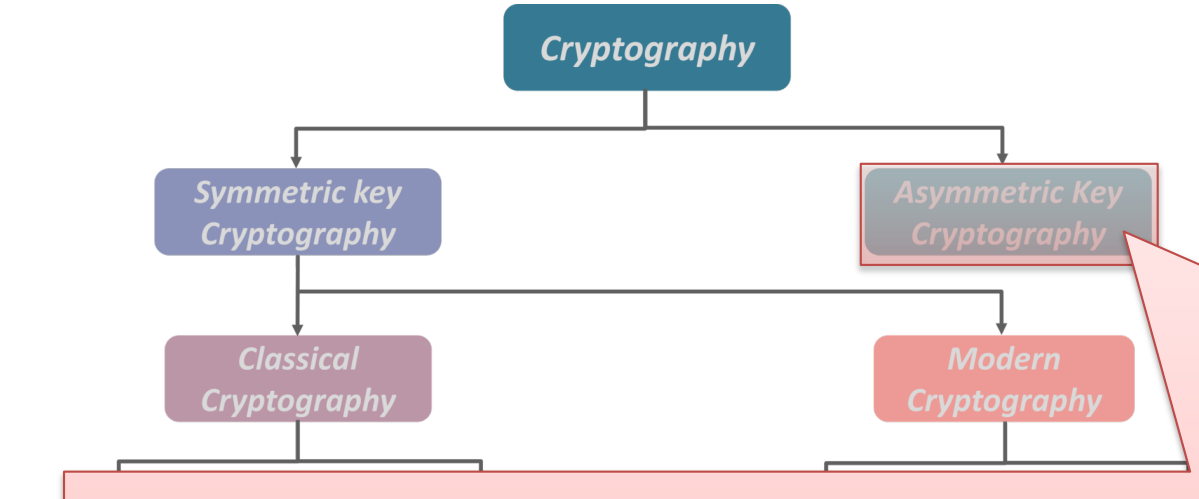


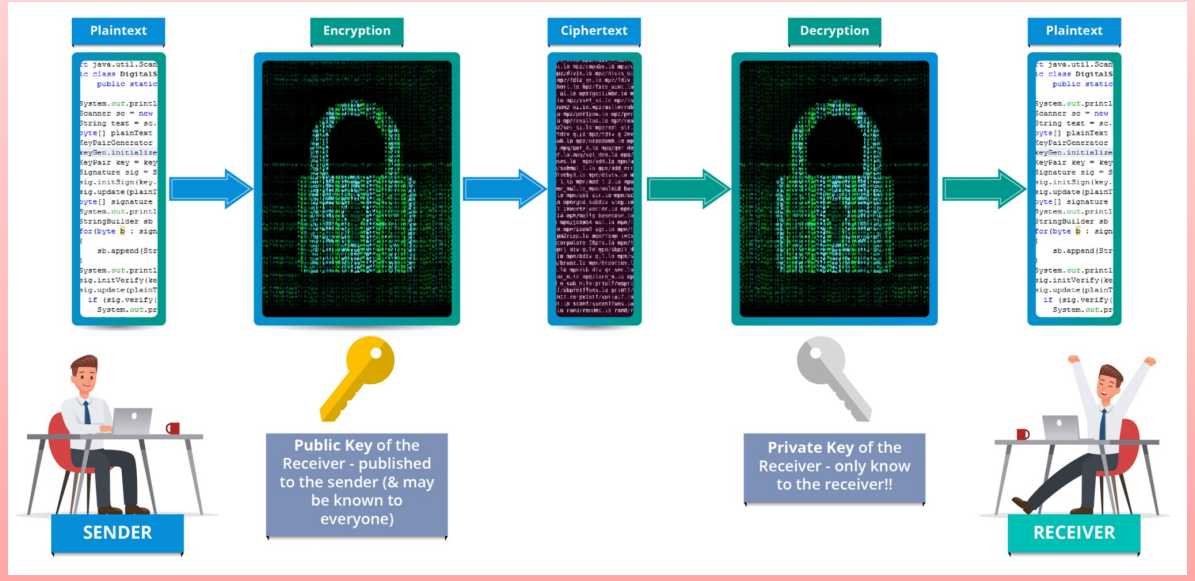




A deterministic algorithm along with a symmetric key is applied to encrypt a block of text, rather than one bit at a time.

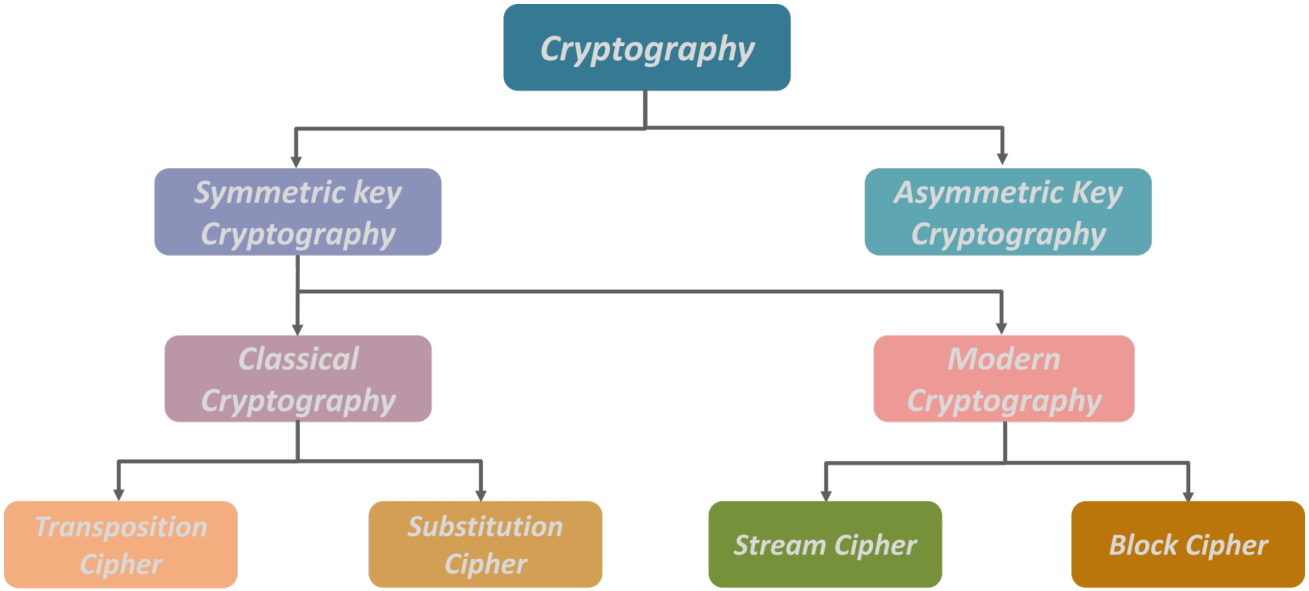
::.. Cryptography (2/5)





Two Keys are used in each operation and are different but are mathematically related, such that retrieving the plain text by decrypting ciphertext is feasible.

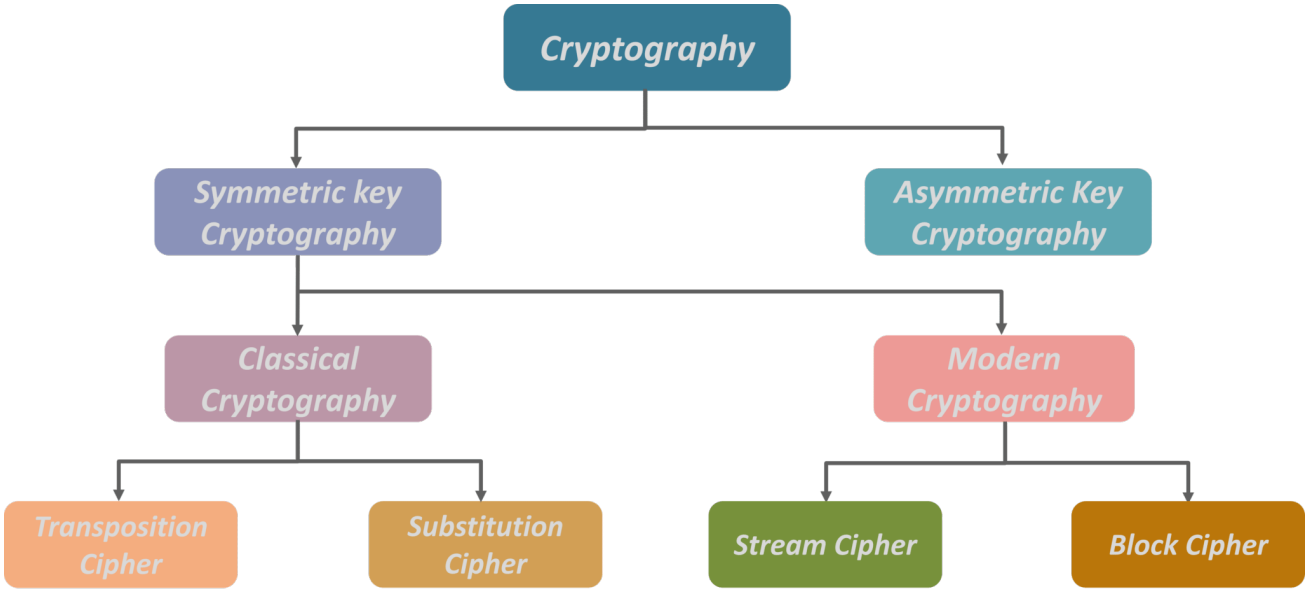
::.. Cryptography (2/5)



The Advanced Encryption Standard (AES) is a specification for the encryption of electronic data and belongs to the block ciphers.

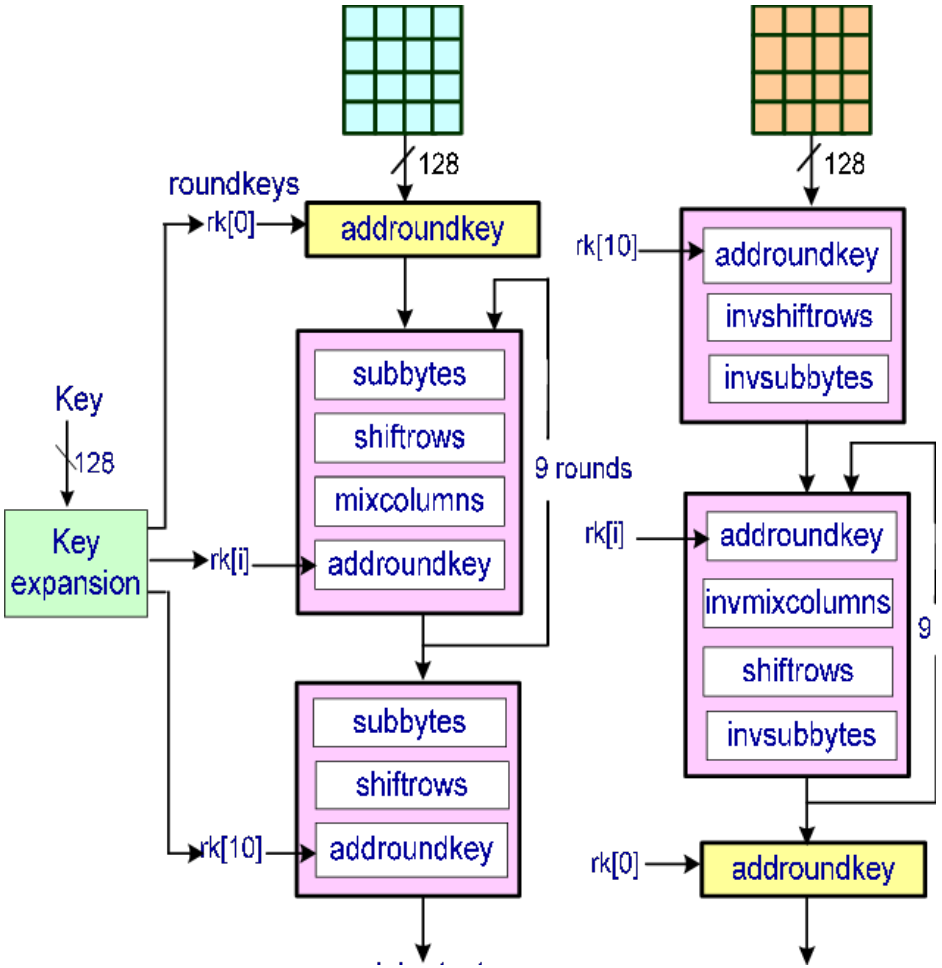
AES is based on a design principle known as a substitution–permutation network, with a fixed block size of 128 bits, and a key size of 128, 192, or 256 bits.

::.. Cryptography (2/5)



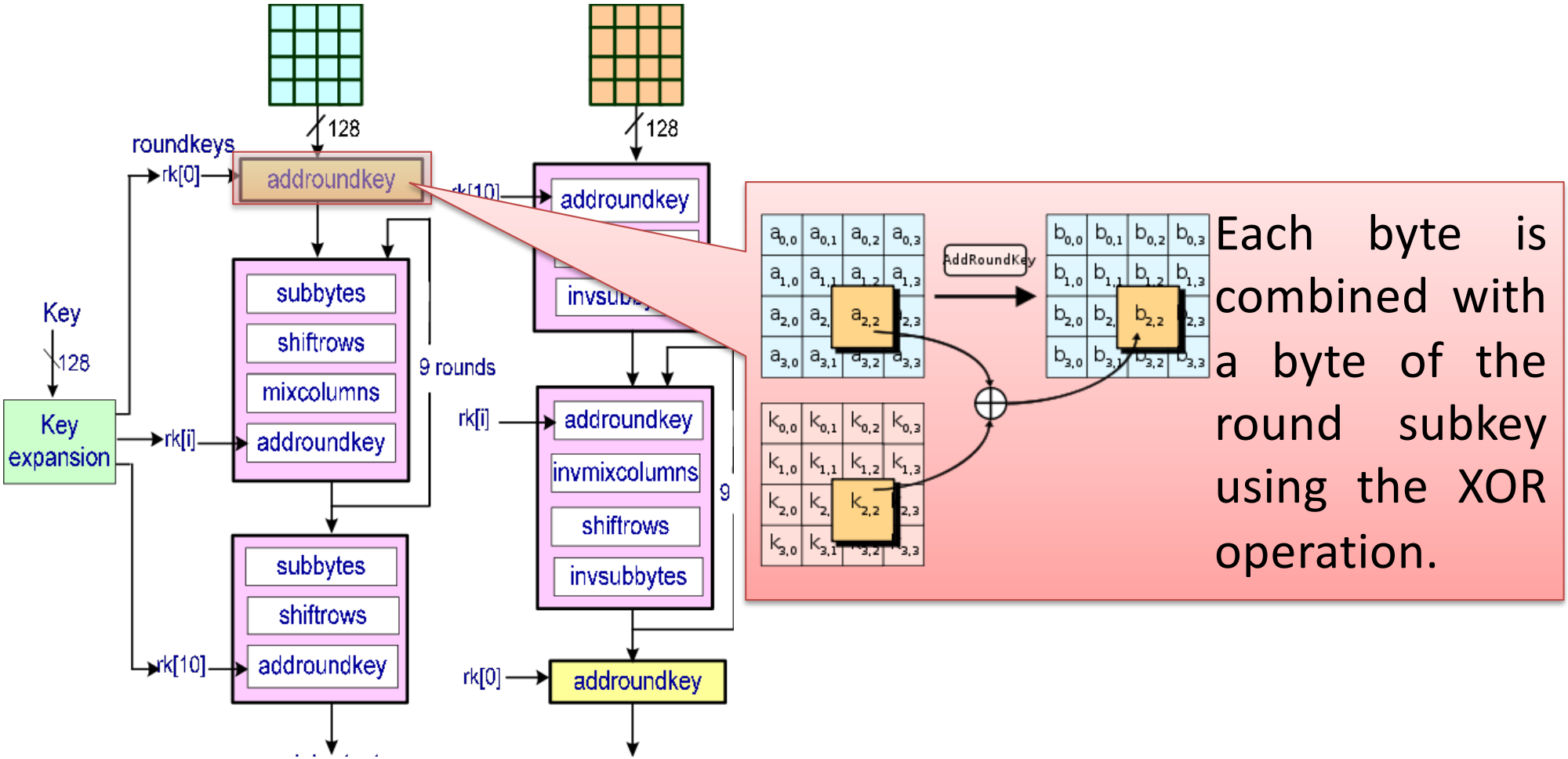
Rivest Cipher 4 (RC4) is the most widely used of all stream ciphers, particularly in software in various protocols like WEP and WPA.

::.. Cryptography (3/5)



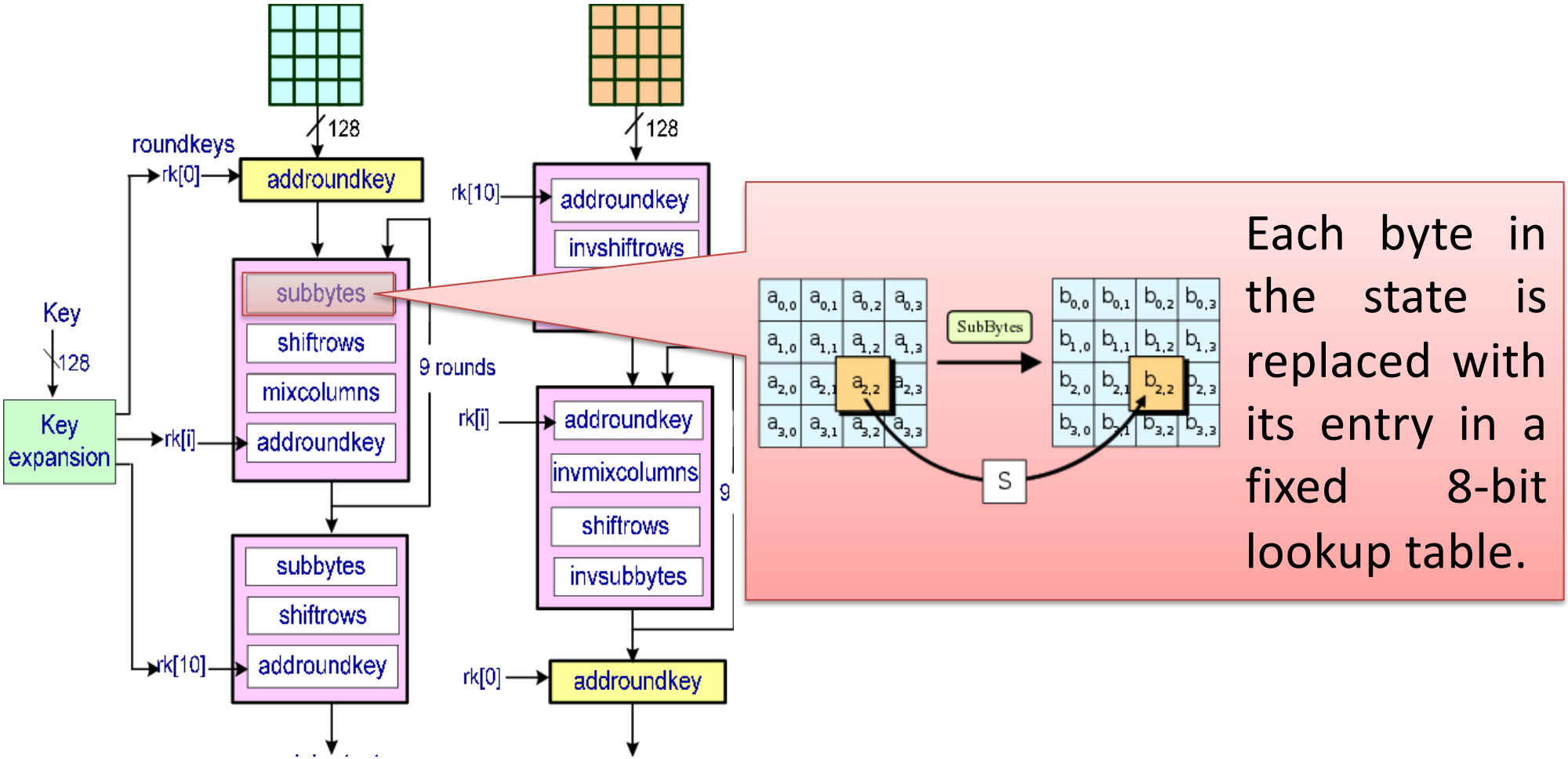
The AES implementation

::.. Cryptography (3/5)



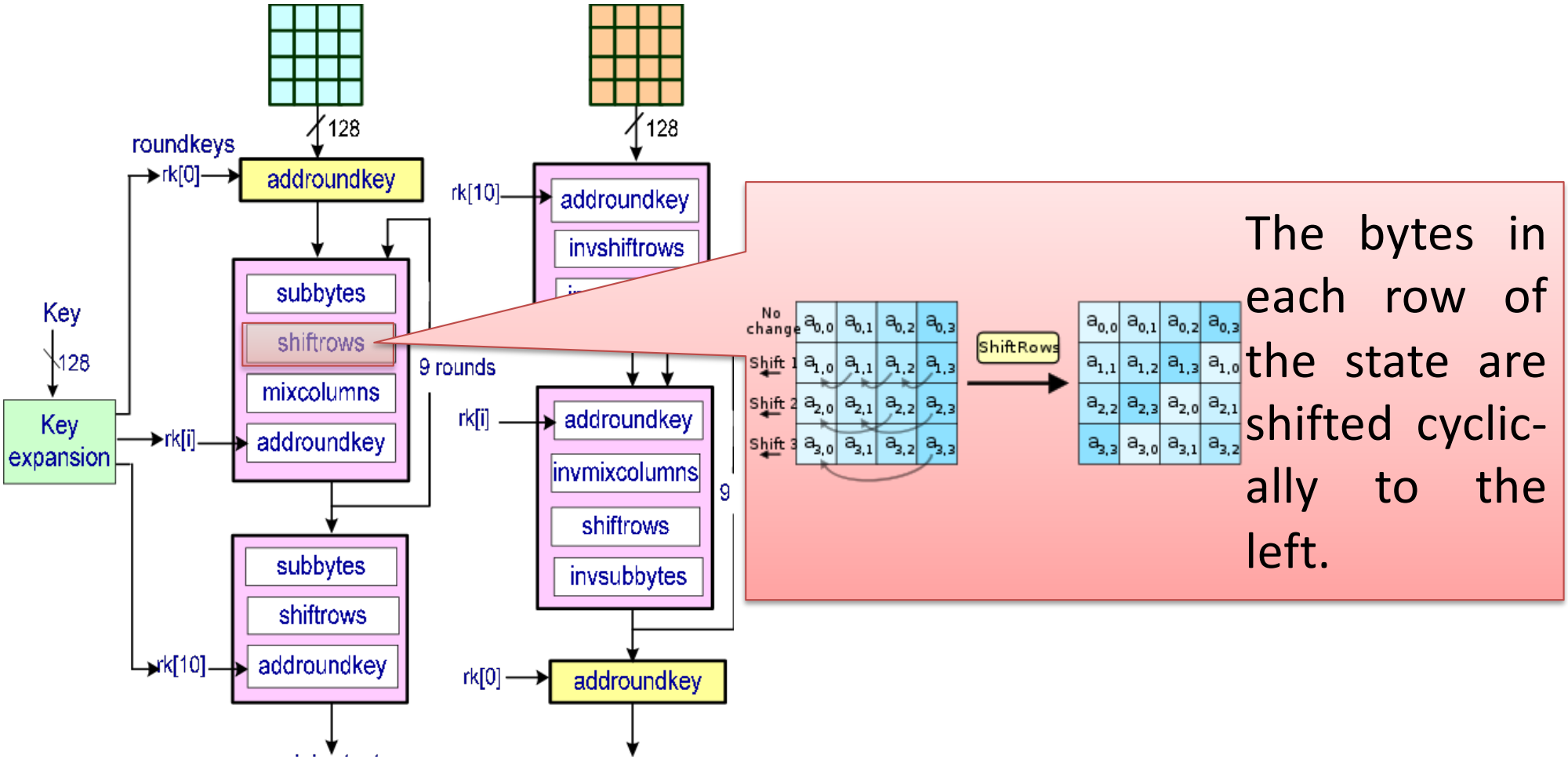
The AES implementation

::.. Cryptography (3/5)



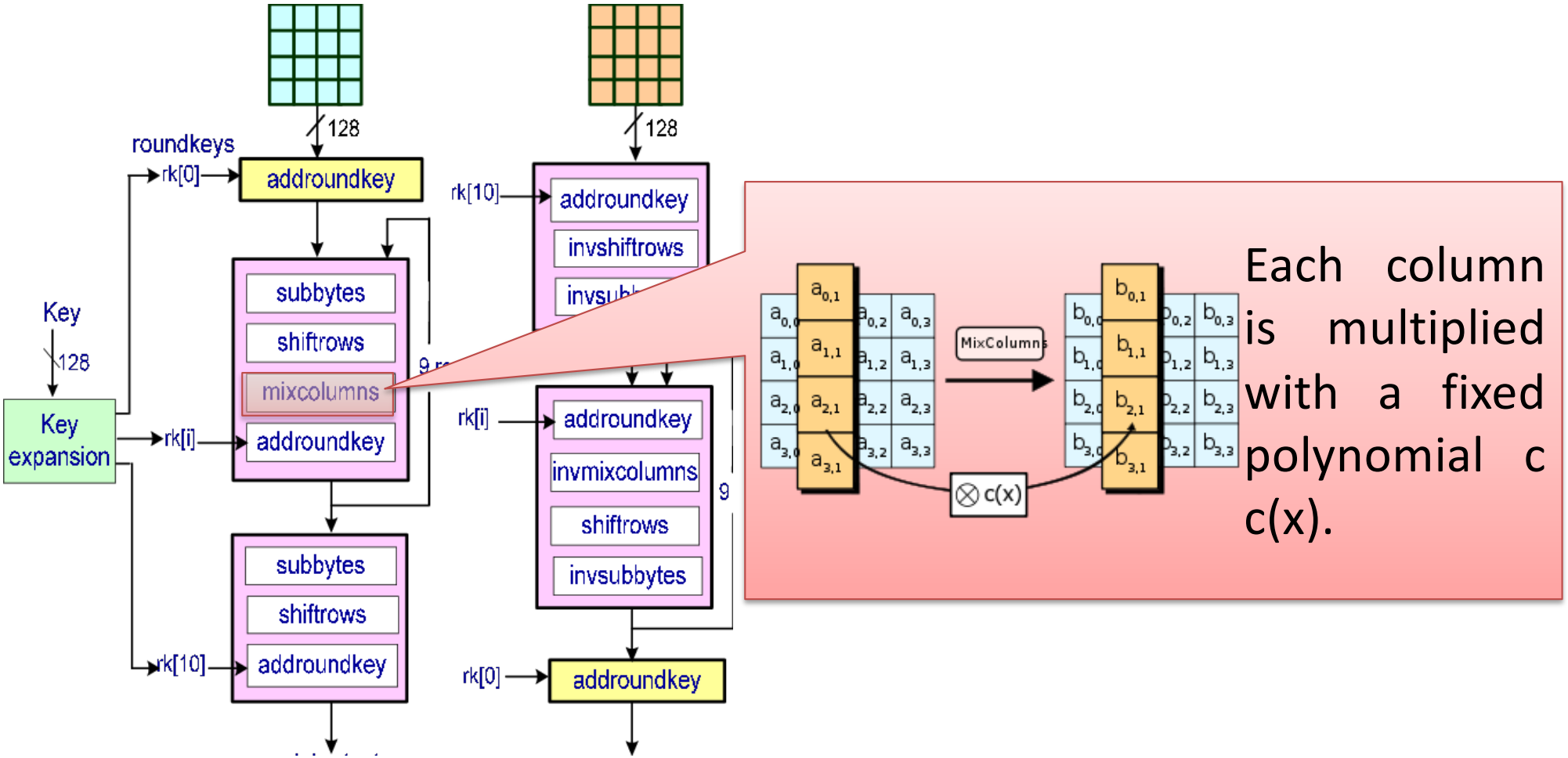
The AES implementation

::.. Cryptography (3/5)



The AES implementation

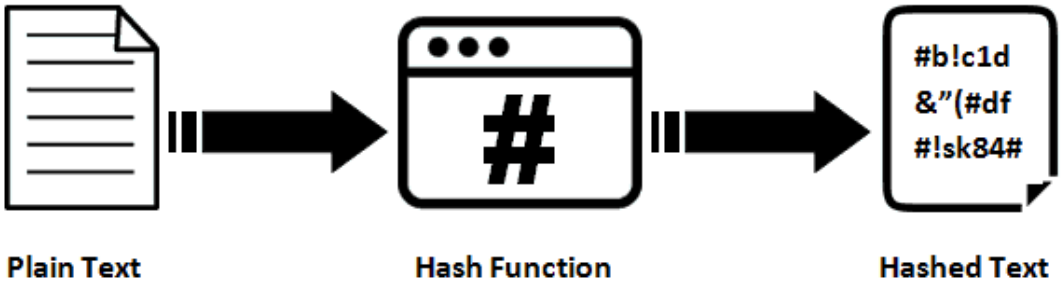
::.. Cryptography (3/5)

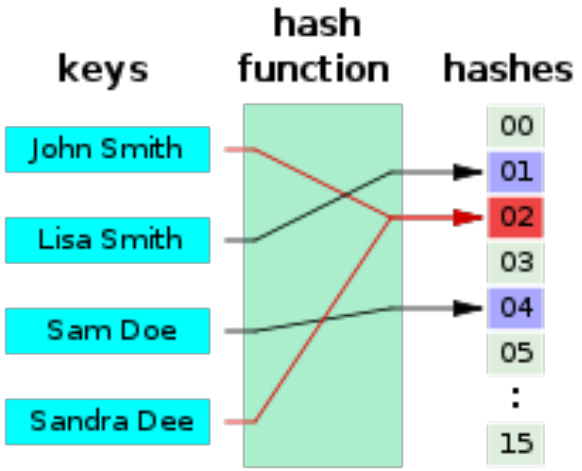


The AES implementation

::.. Cryptography (4/5)

Hashing is the process of converting a given piece of data of arbitrary size into another fixed-size value by using a proper algorithm, which should be one-way so that the hash cannot be converted back into the original piece of data.

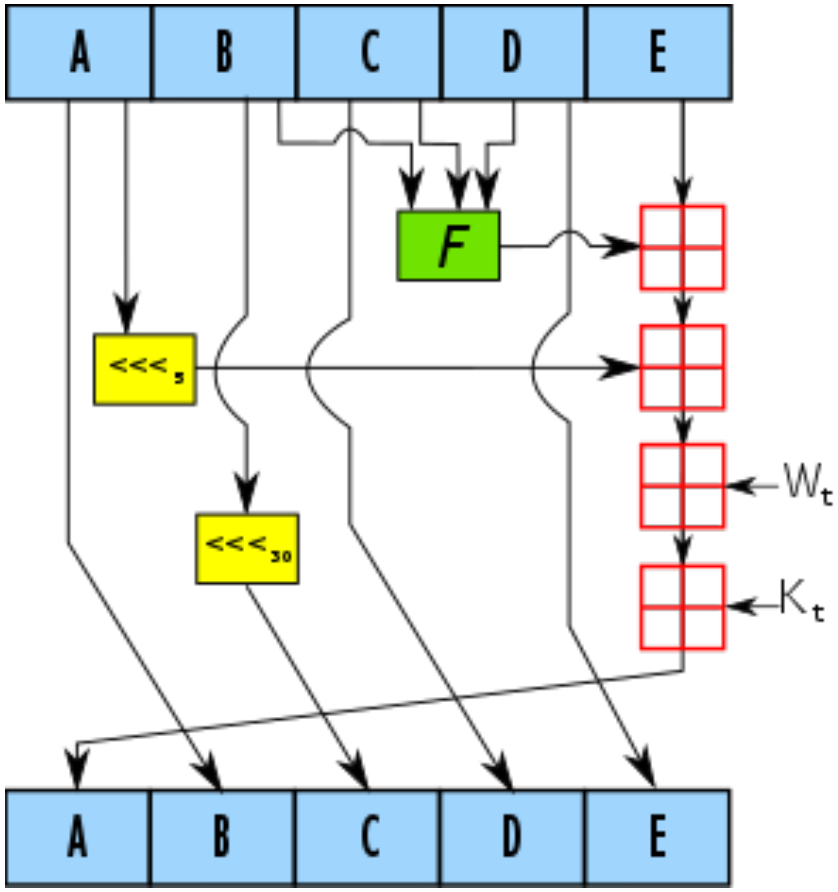




The Secure Hash Algorithms (SHA) are a family of cryptographic hash functions. SHA-1 is the most used algorithm and forms the basis of many applications and protocols, including TLS and SSL, SSH, and IPsec, or for digital signing of documents.

::.. Cryptography (5/5)

SHA-1 produces a message digest of 160 bits out of the input, (of maximum 2^{64}-1 bits) as composed of 512-bit blocks. It consists of a series of operations in a round, for a total of 80 rounds.

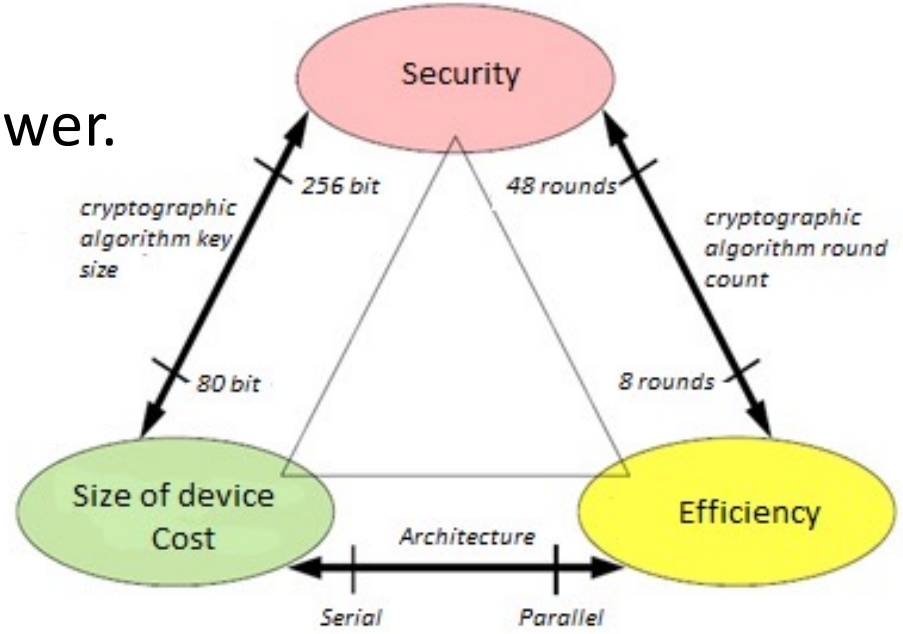


- A, B, C, D and E are 32-bit words of the state, with an initial value;
- F is a nonlinear function;
- $\lll_n$ denotes a left bit rotation by n places, where n varies;
- W_t is the expanded message word of round t;
- K_t is the constant of round t;
- The red square denotes addition modulo 2^{32}.

::.. Cryptography In IoT - Limits

While AES and SHA work well together within computer systems, they struggle in an Internet of Things world as they take up:

- too much processing power;
- too much physical space;
- too much consumed battery power.



As a rule, any two of the three design goals can be easily achieved, while meeting all three requirements - an extremely difficult task.

::.. Cryptography In IoT - Limits

While AES and SHA work well together within computer systems, they struggle in an Internet of Things world as they take up:

- too much processing power;
- too much physical space;
- too much consumed battery power.

Both the national and international organisations outline a number of methods which can be used for lightweight cryptography and which could be useful in IoT and RFID devices. They define the device spectrum as follows:

- Conventional cryptography - Servers and Desktops; Tablets and smart phones.
- Lightweight cryptography - Embedded Systems; RFID and Sensor Networks.

Introduction to Lightweight Cryptography

::.. Lightweight Cryptography (1/3)

In lightweight cryptography, smaller block size (typically 64 bits or 80 bits), smaller keys (often less than 90 bits) and less complex rounds (and where the S-boxes often just have 4-bits) are often seen. Along with this, it has been identified as traditional methods have weaknesses against side channel attacks.

For lightweight cryptography, the main constraints that we have are typically related to power requirements, gate equivalents (GEs) and timing. Often lightweight cryptography methods balance performance (throughput) against power drain and GE and do not perform as well as main-stream cryptography standards (such as AES and SHA-256).

::.. Lightweight Cryptography (2/3)

The lightweight cryptography methods must also have a low requirement for RAM (where the method requires the usage of running memory to perform its operation) and ROM (where the method is stored on the device).

In the IoT, many interconnected resource-constrained devices are not designed to carry out expensive conventional cryptographic computation, which makes it difficult to implement sufficient cryptographic functions.

A number of lightweight cryptography solutions have been standardised as the ISO/ICE 29121. For secure communication, the lightweight primitives have been embedded into existing protocols, such as IPSec and TLS, and some embedded libraries have been released, such as wolfSSL, sharkSSL, etc.

::.. Lightweight Cryptography (2/3)

The lightweight cryptography methods must also have a low requirement for RAM (where the method requires the usage of runni
meth

In th
are
crypt
imple

A nu
stand
the l
proto
have been released, such as wolfSSL, sharkSSL, etc.

wolfSSL (https://www.wolfssl.com/5) is a small, portable, embedded SSL/TLS library targeted for use by embedded systems developers. It is an open source implementation of TLS written in the C programming language. It includes SSL/TLS client libraries and an SSL/TLS server implementation as well as support for multiple APIs, including an OpenSSL compatibility interface with the most commonly used OpenSSL functions.
By default, wolfSSL uses the cryptographic services provided by wolfCrypt, a back-end crypto implementation for several popular software packages and libraries.

::.. Lightweight Cryptography (2/3)

The lightweight cryptography methods must also have a low requirement for RAM (where the method requires the usage of runni
meth

In th
are
crypt
imple

A nu
stand
the l
proto
have been released, such as wolfSSL, sharkSSL, etc.

SharkSSL (https://realtimelogic.com/products/sharkssl/5) is the smallest, fastest, and best performing embedded TLS v1.0/1.1/1.2 solution. With its array of compile-time options, the small and fast SharkSSL can be fine-tuned to a light footprint that occupies less than 20kB, while maintaining full x.509 authentication, using industry standard encryption.
It uses the RayCrypto engine, which is an extremely small and fast embedded crypto library designed specifically for embedded resource-constrained devices, including bare metal and RTOS environments. RayCrypto is written in ANSI C and Assembly Optimizations are available.

::.. Lightweight Cryptography (3/3)

More generally and regardless of the platform, an implementation is a trade-off between security, performances and cost.

- A cheap and efficient chip may be vulnerable to side-channel attacks;
- a cheap side-channel-resilient one may be slow;
- a fast and secure one can be expected to be expensive.

The role of lightweight algorithms is to provide better trade-offs in this space by e.g. lowering the difficulty of a very efficient hardware implementation.

::.. Hardware Implementation (1/5)

If the primitive is implemented in hardware, the following metrics describe the efficiency of the implementation.

- The memory consumption and the implementation size are lumped together into its gate area which is measured in Gate Equivalents (GE). It quantifies how large the physical area needed for a circuit implementing the primitive is. The lower it is, the better.
- The throughput, measured in bits or bytes per second, corresponds to the amount of plaintext processed per time unit. The higher it is, the better.
- The latency, measured in seconds, correspond to the time taken to obtain the output of the circuit once its input has been set. The lower it is, the better.

::.. Hardware Implementation (1/5)

If the primitive is implemented in hardware, the following metrics describe the efficiency of the implementation.

- The memory consumption and the implementation size are lumped together into its gate area which is measured in Gate Equivalents (GE). I
needed for a ci
it is, the better.
- The throughput,
corresponds to the
unit. The higher it is
- The latency, measu
taken to obtain th
been set. The lower

This value defines the complexity of the production technology, regardless of the complexity of digital electronic circuits.

Element	Technological process, нм	Area, нм^2	GE
NOT	0,18	6,451	0,67
NOT-AND	0,18	9,677	1
NOT-OR	0,18	9,677	1
AND	0,18	12,902	1,33
OR	0,18	12,902	2,33
Multiplexor	0,18	22,579	2,67
Modulo 2	0,18	25,805	4,67
Modulo 3	0,18	45,158	5,33

::.. Hardware Implementation (2/5)

- The power consumption, measured in Watts, quantifies the amount of power needed to use the circuit. The lower it is, the better.

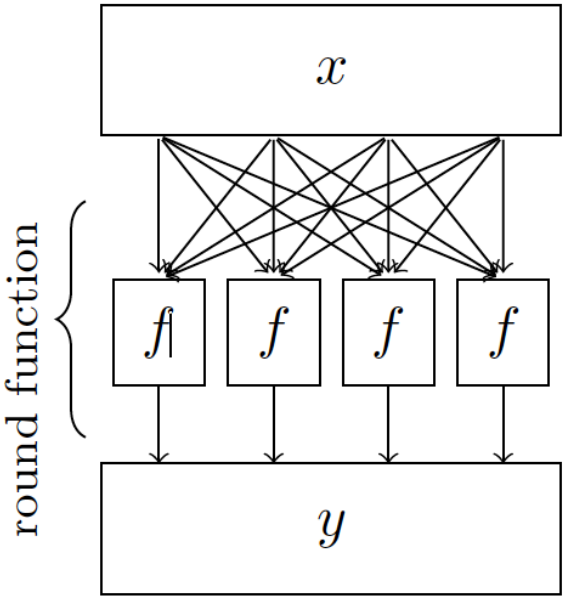
These four criteria compete with one another.

- A low latency tends to imply a higher area. Different implementations of the AES have been compared and the by far smallest implementation is also by far the slowest while the most energy efficient are the largest.

The optimal trade-off between these quantities is very much context dependent. In fact, primitives have been proposed that have been optimized for different corners of the design space: some allow a very low latency implementation, others a very small one (in terms of GE), etc.

::.. Hardware Implementation (3/5)

Regardless of the exact platform, a given primitive may be implemented using different approaches.



Round-based ($t = 1$).

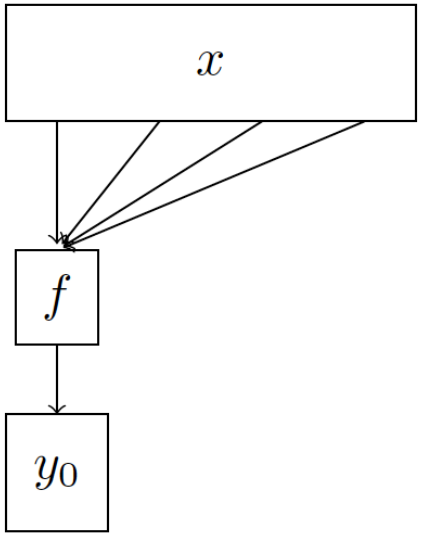
Let us consider round-based permutations. A direct approach consists in storing the full internal state (along with the key state, if any) and then perform one round using a circuit operating on the full state at once.

This is called a **round-based implementation**. In this case, each output bit is evaluated in parallel.

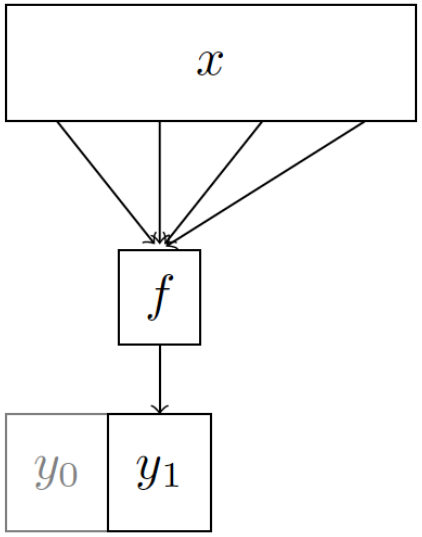
It is also possible to push this logic further and, rather than doing rounds one after another, to do several at once instead. Such unrolled implementations are popular when a low-latency is targeted as those allow a full evaluation in one clock cycle.

::.. Hardware Implementation (4/5)

The downside is then the far larger size of the circuit. Serial implementations work differently.



Serial $(t = 1)$.

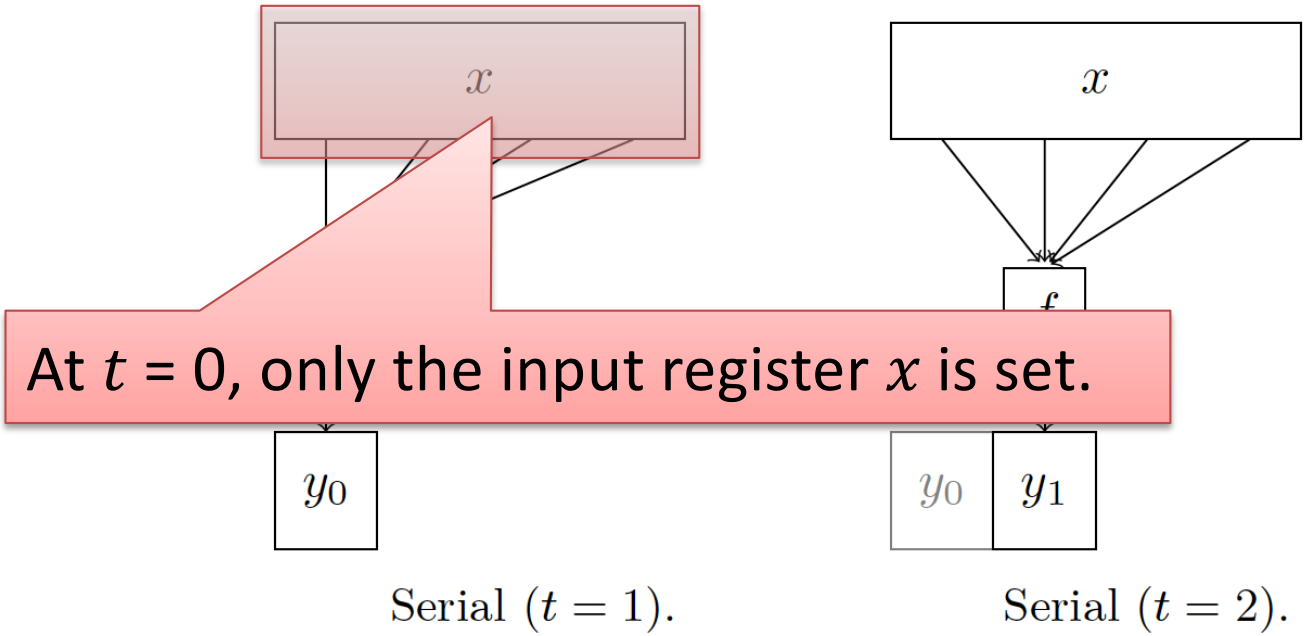


Serial $(t = 2)$.

They only update a small part of the state at a time—typically the size of an S-Box output.

::.. Hardware Implementation (4/5)

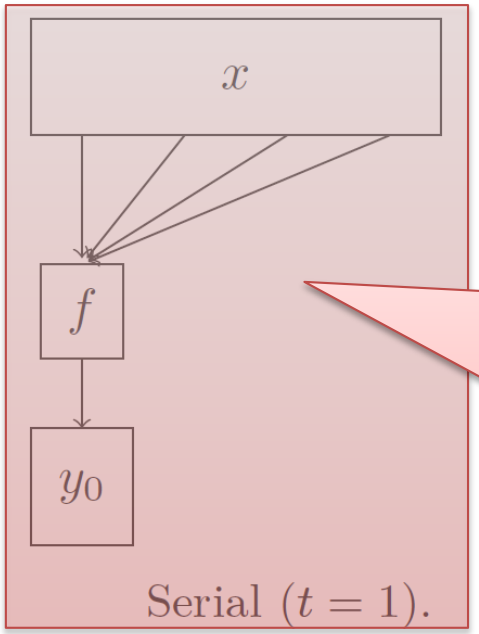
The downside is then the far larger size of the circuit. Serial implementations work differently.

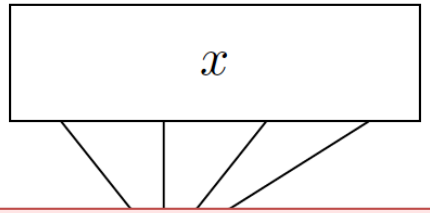


They only update a small part of the state at a time—typically the size of an S-Box output.

::.. Hardware Implementation (4/5)

The downside is then the far larger size of the circuit. Serial implementations work differently.



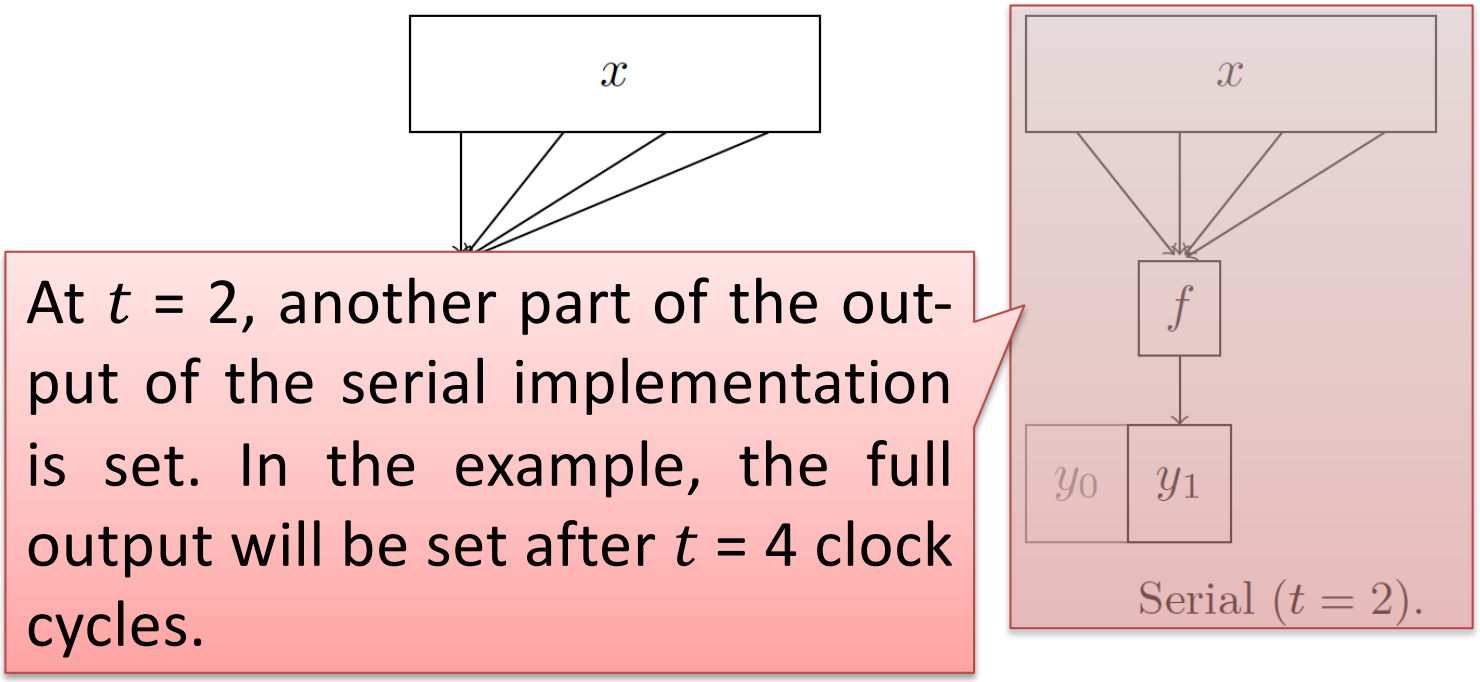


At t = 1, after the first clock cycle, the whole state of the parallel implementation is set but only a part of the output of the serial implementation is generated.

They only update a small part of the state at a time—typically the size of an S-Box output.

::.. Hardware Implementation (4/5)

The downside is then the far larger size of the circuit. Serial implementations work differently.

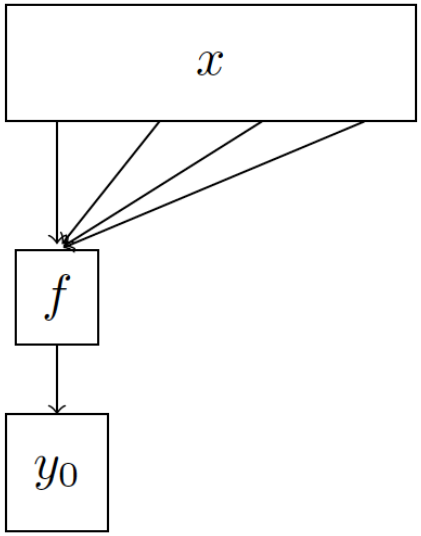


At t = 2, another part of the output of the serial implementation is set. In the example, the full output will be set after t = 4 clock cycles.

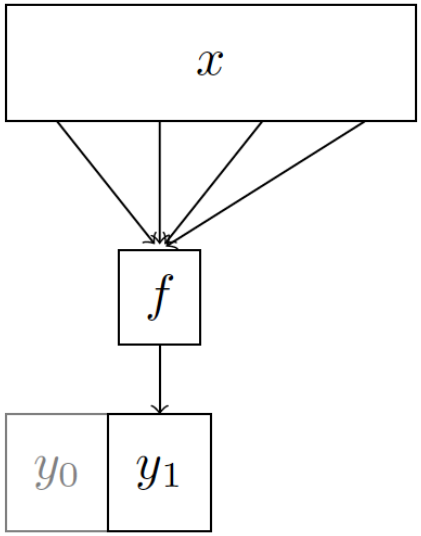
They only update a small part of the state at a time—typically the size of an S-Box output.

::.. Hardware Implementation (4/5)

The downside is then the far larger size of the circuit. Serial implementations work differently.



Serial $(t = 1)$.



Serial $(t = 2)$.

Due to the simplicity of the circuit logic involved, serialized implementations allow a far higher clock frequency so they may not be as slow as one might fear. Still, their main advantage is a much smaller circuit.

::.. Hardware Implementation (5/5)

Memory is often the most expensive part of the implementation of a lightweight primitive. As a consequence, it is preferable to operate on small blocks using a small key.

However, some space can be saved by hard-coding or “burning” the keys into the device. That is, instead of using read/write memory to store the key, use read-only structures. In order for this method to be viable, the key schedule must build the round keys using only simple operations taking as input the bits of the master key. In particular, no key state can be operated upon.

Energy and power efficiency of the hardware implementation are at the core of the design of the Midori block cipher. A very low latency demands specific design choices as illustrated by the lightweight block ciphers PRINCE followed by Mantis and Qarma.

::.. Software Implementation (1/5)

Lightweight primitives can be also implemented in software, typically for use on microcontrollers.

In this case, the relevant metrics are the RAM consumption, the code size and the throughput:

- RAM consumption corresponds to the amount of data which is written to memory during each evaluation of the function,
- Code size is the fixed amount of data which is needed to evaluate the function independently from its input, and
- the throughput measures the average quantity of data which is processed during each clock cycle.

The first two are measured in bytes and should be minimized while the latter is measured in bytes per cycle and should be maximized. Again, these three quantities are not independent.

::.. Software Implementation (2/5)

For example, loading information from the RAM into CPU registers is a costly operation and so is its inverse. Therefore, limiting the number of such operations leads to a decrease in both RAM consumption and an increase in throughput.

Micro-controllers operate on small words of 8, 16 or 32 bits, so that some operations can have counter intuitive costs.

- On most 32-bit processors, all 32-bit rotations have the same (low) cost.
- However, 8-bit micro-controllers do not share this property. Rotations by multiples of 8 are quite cheap as they merely correspond to shuffling 8-bit registers. On the other hand, there is no dedicated instruction to perform other rotations 32-bit rotations.

::.. Software Implementation (3/5)

As a consequence, rotations have very different and non-negligible prices.

It is also quite different from the hardware case, where all bit permutations are extremely cheap.

Much like in the hardware case, there are different trade-offs and implementation techniques. For example, the subkeys of a block cipher can be precomputed to save time at the cost of memory during an encryption. It is also possible to use much more sophisticated techniques like bit-slicing which, while they may not always be applicable, can lead to substantial performance gains.

::.. Software Implementation (3/5)

It is a technique for constructing a processor from modules of processors of smaller bit width, for the purpose of increasing the word length at a more reasonable cost, using off-the-shelf components that could be custom-configured.

The basic consists of:

1. expressing the cipher in terms of single-bit logical operations (AND, OR, XOR, NOT, etc.), as if you were implementing it in hardware, and
2. carrying out those operations for multiple instances of the cipher in parallel, using bitwise operations on a CPU.

more sophisticated techniques like bit-slicing which, while they may not always be applicable, can lead to substantial performance gains.

::.. Software Implementation (4/5)

Given a scalar processor with a w-bit word size, let x_i denote the i-th bit of a machine word x, where i is the index of the bit.

Such a processor operates natively on word-sized operands.

- For example, with a single operation one might perform addition of w-bit operands x and y to produce r = x + y, or component-wise XOR to producer $i=x_i \oplus y_i$ for all $0 \leq i < w$.

This ability is restricted when an algorithm is required to perform some operations involving different bits from the same word.

- For example, one might be required to combine x_i and x_j, where i != j, using an XOR in order to compute the parity of x. In this situation one is required to shift (and potentially mask) the bits so they are aligned at the same index ready for combination through a native, component-wise XOR.

::.. Software Implementation (5/5)

The technique of bit-slicing offers a way to reduce the associated overhead. Instead of representing the w-bit value x as one machine word, they are represented by using w machine words where word i contains x_i aligned at the same fixed index j.

As such, there is no need to align bits ready for use in a component-wise XOR operation. Additionally, since native word-oriented logical operations in the processor operate on all w bits in parallel, one can pack w different values into the w words and proceed using an analogy of a SIMD-style parallelism, ending up with a net gain in throughput.

Conversion to and from a bit-sliced representation can represent an overhead but this can be amortised if the cost of computation using the bit-sliced values is significant enough.

::.. SCA Robustness (1/3)

Side-channel attacks use some special knowledge about the implementation of a cipher to break its security. For example, observing the power consumption of an encryption can leak insights about the Hamming weight of the output of an S-Box.

Lightweight algorithms are often built so as to decrease the vulnerability of their implementation to such attacks.

SCAs exploit the fact that the different operations performed during an encryption correspond to physical processes which can leak information about a secret such as the key. As a consequence, measuring precisely the power consumption of this device may reveal some information about the input of the S-Box and, thus, about the internal state of the cipher. Both hardware and software implementation can be vulnerable.

::.. SCA Robustness (2/3)

To safeguard against such an attack, all operations can be masked by using an external source of random data to randomize the input of the different operations that might leak information.

More formally, it ensures that the input of each operation is statistically independent from secret data such as the internal state of a block cipher.

- To this aim DES has been modified where the 64-bit input block x is replaced by two blocks x' and x'' such that $x = x' \oplus x''$. This property is preserved by the bit permutations used by this cipher. Each of the eight different 6-to-4 bits S-Boxes S_i is replaced by a pair of 12-to-4 bits functions A_i, f_i, where $f_i(v', v'') = S_i(v' \oplus v'') \oplus A_i(v', v'')$.

::.. SCA Robustness (3/3)

- When an S-Box input v is split into $v = v' \oplus v''$, such functions verify $f_i(v', v'') \oplus A_i(v', v'') = S_i(v)$, meaning that they can be used to properly evaluate S_i while only evaluating functions with randomized inputs.

Different operations can be more or less easy to mask. Thus, it is possible and indeed becomes a trend in lightweight cryptography to simplify the implementation of such counter-measures by designing a cipher using exclusively such operations.

::.. Lightweight Implementations (1/2)

Many implementers have worked on optimizing the implementation of the AES with some success in both hardware and software.

Some devices are sometimes shipped with a hardware acceleration module providing AES encryption and decryption, effectively adding a new set of instructions dedicated entirely to a quick evaluation of this block cipher.

Lightweight symmetric algorithms may seem unnecessary. However, block cipher hardware acceleration has its limitations.

- The hardware-accelerated encryption used by many devices are vulnerable to various forms of side-channel attacks.

::.. Lightweight Implementations (2/2)

Without hardware support, the software-optimized AES is rather fast. However, its implementation requires storing at least the full look-up-table of its 8-bit S-Box or the full code for its bit-slice implementation, neither of which can be made very small.

While the AES is a reasonably lightweight block cipher, its large S-Box, large block size and inherent vulnerability to SCA caused by its look-up-based S-Box mean that there are extreme design goals that it is unlikely to fulfil, a very small code size or a very efficient masked implementation.

Another case where dedicated lightweight algorithms are needed is for hashing. Indeed, standard hash functions need large amounts of memory to store both their internal states and the block they are operating on.

::.. Trends in LC (1/13)

Lightweightness can be seen as a set of specific design constraints. These are tackled differently by different algorithms but some trends emerge when we look at the evolution of lightweight block ciphers.

Non-linearity is a necessary property of any cryptographic primitive. It can be provided by S-Boxes…

::.. Trends in LC (1/13)

Lightweightness can be seen as a set of specific design constraints. These are tackled differently by different algorithms

S-box (substitution-box) is a basic component performing substitution. They are typically used to obscure the relationship between the key and the ciphertext — Shannon's property of confusion.

ok at the evolution of

Non-linearity is a ... property of any cryptographic primitive. It can be provided by S-Boxes...

S_5		Middle 4 bits of input															
		0000	**0001**	**0010**	**0011**	**0100**	**0101**	**0110**	**0111**	**1000**	**1001**	**1010**	**1011**	**1100**	**1101**	**1110**	**1111**
Outer bits	**00**	0010	1100	0100	0001	0111	1010	1011	0110	1000	0101	0011	1111	1101	0000	1110	1001
	01	1110	1011	0010	1100	0100	0111	1101	0001	0101	0000	1111	1010	0011	1001	1000	0110
	10	0100	0010	0001	1011	1010	1101	0111	1000	1111	1001	1100	0101	0110	0011	0000	1110
	11	1011	1000	1100	0111	0001	1110	0010	1101	0110	1111	0000	1001	1010	0100	0101	0011

a fixed table is the S-box from DES (S_5), mapping 6-bit input into a 4-bit output.

::.. Trends in LC (1/13)

Lightweightness can be seen as a set of specific design constraints. These are tackled differently by different algorithms but some trends emerge when we look at the evolution of lightweight block ciphers.

Non-linearity is a necessary property of any cryptographic primitive. It can be provided by S-Boxes or through the use of non-linear arithmetic operations or "bit-sliced S-Box" using basic bitwise operations such as XOR, AND and OR.

An advantage of S-Boxes is the simple security argument based for example on the wide trail strategy that they allow.

::.. Trends in LC (2/13)

S-Box-based algorithms can further be divided into two categories. The first specifies them using Look-Up Tables (LUT) and the second uses bit-sliced algorithms.

::.. Trends in LC (2/13)

S-Box-based algorithms can further be divided into two categories. The first specifies them using Look-Up Tables (LUT) and the second uses bit-sliced algorithms.

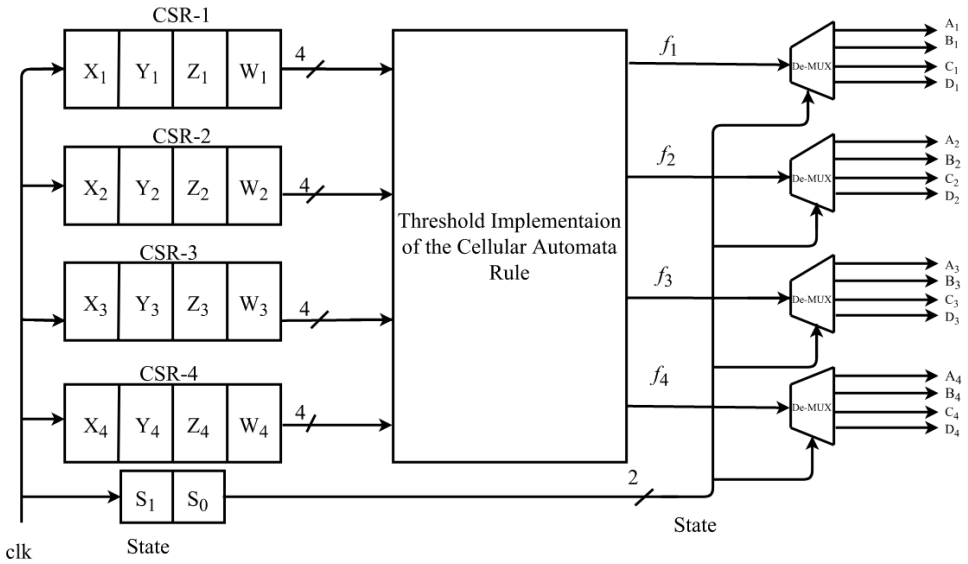
LUT-based algorithms use S-Boxes which are specified via their look-up tables and which are usually implemented via this method in software.

Such functions are useful as they can offer (near) optimal cryptographic properties using what is essentially a unique operation. However, such an implementation requires storing all possible outputs which, for an 8-bit S-Box such as the one used by the AES, has a significant cost.

The table look-up is the operation leaking the most information.

::.. Trends in LC (3/13)

S-Boxes intended to be implemented using LUTs in software usually correspond to a simple electronic circuit which can be efficiently implemented in hardware. In the case of hardware, it is possible to significantly reduce the cost of the S-Box.



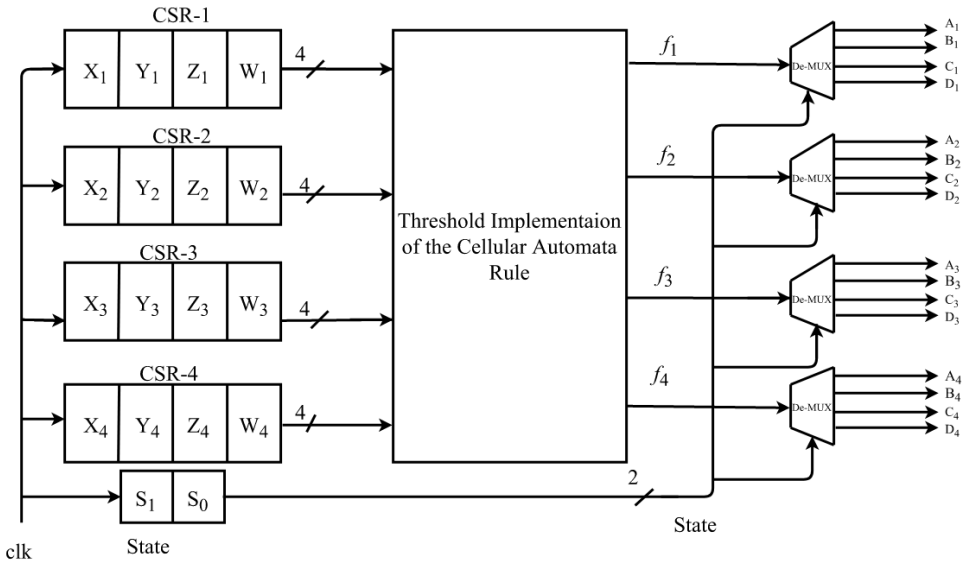
CSR: Cyclic Shift Register
State: 2-Bit Counter
clk: Clock Signal

S-Box Inputs: (X,Y,Z,W)
Share-1: (X_1,Y_1,Z_1,W_1)
Share-2: (X_2,Y_2,Z_2,W_2)
Share-3: (X_3,Y_3,Z_3,W_3)
Share-4: (X_4,Y_4,Z_4,W_4)

S-Box Outputs: (A,B,C,D)
Share-1: (A_1,B_1,C_1,D_1)
Share-2: (A_2,B_2,C_2,D_2)
Share-3: (A_3,B_3,C_3,D_3)
Share-4: (A_4,B_4,C_4,D_4)

::.. Trends in LC (3/13)

S-Boxes intended to be implemented using LUTs in software usually correspond to a simple electronic circuit which can be efficiently implemented in hardware. In the case of hardware, it is possible to significantly reduce the cost of the S-Box.



CSR: Cyclic Shift Register
State: 2-Bit Counter
clk: Clock Signal

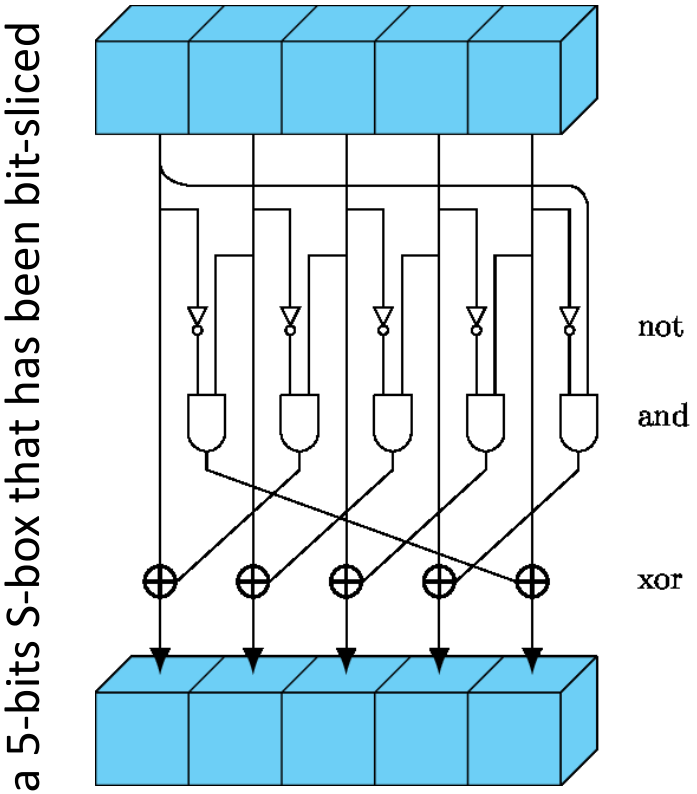
S-Box Inputs: (X,Y,Z,W)
Share-1: (X_1,Y_1,Z_1,W_1)
Share-2: (X_2,Y_2,Z_2,W_2)
Share-3: (X_3,Y_3,Z_3,W_3)
Share-4: (X_4,Y_4,Z_4,W_4)

S-Box Outputs: (A,B,C,D)
Share-1: (A_1,B_1,C_1,D_1)
Share-2: (A_2,B_2,C_2,D_2)
Share-3: (A_3,B_3,C_3,D_3)
Share-4: (A_4,B_4,C_4,D_4)

Bit-slice-based algorithms also use S-Boxes but, in this case, it is intended from the start to be implemented in a bit-sliced fashion: no table look-ups are required but some bitwise operations such as AND and XOR are performed.

::.. Trends in LC (4/13)

S-Boxes specified via a bit-sliced representation require only a limited number of logical operations to be evaluated: 4-bit ones usually need only 4 non-linear gates during their evaluation which makes their software implementation particularly easy to mask.



A simple bit-sliced implementation is also related to a small area for a hardware implementation, meaning that such algorithms can be expected to perform well in hardware as well.

Bit-sliced S-Boxes are a popular choice for the design of lightweight algorithms.

::.. Trends in LC (5/13)

ARX-based algorithms rely on modular addition to provide non-linearity while word-wise rotations and XOR provide diffusion, hence the name: Addition, Rotation, XOR.

The bits of highest weight in the output of a modular addition are highly non-linear functions due to the propagation of the carry. However, the lower weight bits retain a simple dependency. Furthermore, some differentials and linear approximations have probability 1, meaning that the structure of the linear part must be studied carefully.

Justifying the security of an ARX-based algorithm is far more difficult than for an S-Box-based design because tools such as the wide-trail strategy cannot be applied.

::.. Trends in LC (6/13)

Modular addition is extremely cheap in software. Not only does it consist in few operation, it also uses fewer or no additional registers as it can often be performed in place using the “+=” operator.

As a consequence, ARX-based ciphers are among the best performers for microcontrollers.

::.. Trends in LC (6/13)

Modular addition is extremely cheap in software. Not only does it consist in few operation, it also uses fewer or no additional registers as it can often be performed in place using the “+=” operator.

As a consequence, ARX-based ciphers are among the best performers for microcontrollers.

The choice of the non-linear layer is particularly important when it comes to easing the implementation of counter-measures against side-channel attacks. Indeed, depending on the structure of the non-linear layer, these counter-measure can be more or less costly and have a varying impact on performances.

Popular counter-measure against side-channel attacks are threshold implementations and masking.

::.. Trends in LC (6/13)

Both rely on secret sharing: the idea is to obfuscate the operands of operations whose physical characteristics depend on the value of their input. Both require an external source of random bits which will come with its own cost.

- The cost of a threshold implementation increases with the algebraic degree of the operation considered. As a consequence, the threshold implementation of a large and cryptographically strong S-Box is likely to be prohibitively expensive unless it was designed with a special structure.
- The bit-sliced S-Boxes exhbit good performances in the non-protected case, but they also allow even better side channel-resilient implementations. Since the S-Box is evaluated in a bit-sliced fashion, it is possible to mask each AND separately and to perform the corresponding operations in parallel.

::.. Trends in LC (7/13)

- In order for this method to be efficient, it is necessary that the S-Box uses as few ANDs as possible, which is at odds with its cryptographic strength.

::.. Trends in LC (7/13)

- In order for this method to be efficient, it is necessary that the S-Box uses as few ANDs as possible, which is at odds with its cryptographic strength.

Linear operations provide diffusion, i.e. to ensure that all parts of the state depend on one another after several rounds.

- A Maximum Distance Separable (MDS) matrix M operating on vectors of length d is such that hw(x) + hw($M(x)$) ≤ d + 1 for all non-zero x, where hw(x) counts the number of non-zero elements in the vector x. Such matrices provide optimal diffusion and can be used to prove the resistance against linear and differential attacks of the ciphers built using them. This wide trail argument was most prominently used in AES and was later used by many related algorithms.

::.. Trends in LC (8/13)

While they provide optimal diffusion, such diffusion layers tend to be more expensive than the others. The common trade-off between performance and security level is needed.

Due to their very low cost in hardware, simple bit permutations are a popular choice for algorithms targeting such embedded platforms. The downside is that they are a priori quite expensive in software which is usually ill-suited for efficient bit fiddling.

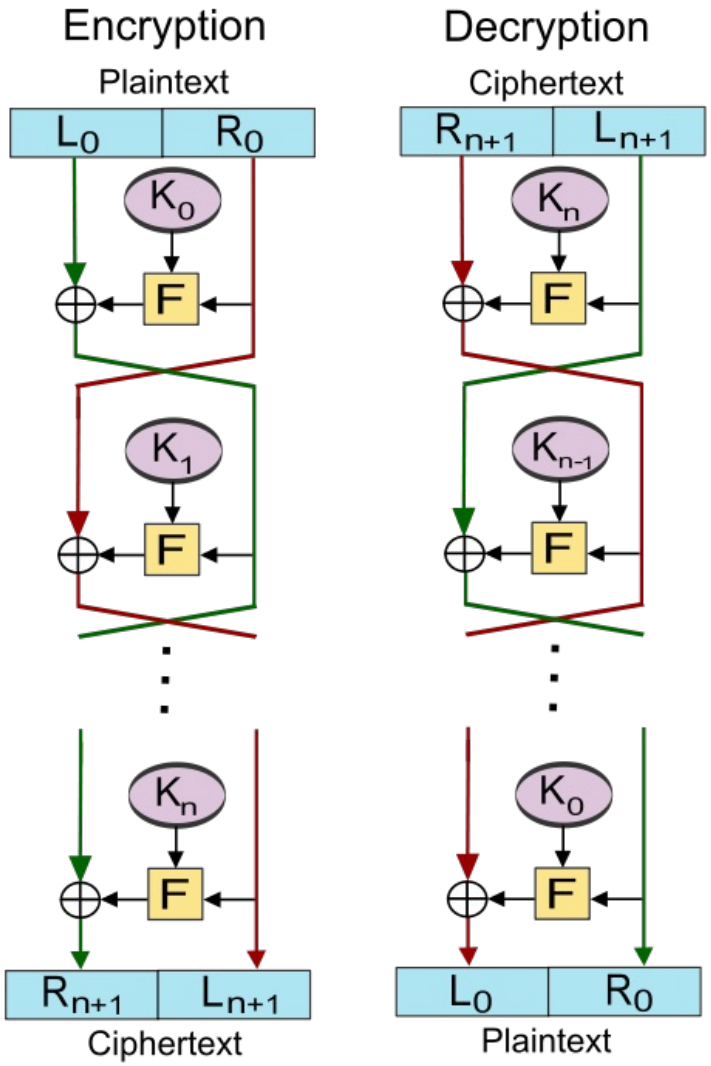
Generalized Feistel Networks (GFN) relying entirely on nibble permutation to provide diffusion between their branches can also be fitted in this category. In their case, a software implementation is a bit cheaper because the bits are grouped into larger groups, meaning that it is not necessary to treat the bits one by one.

::.. Trends in LC (8/13)

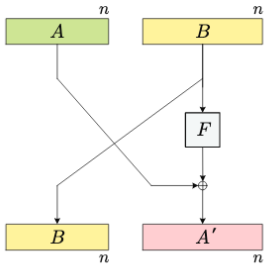
A Feistel cipher is a symmetric structure used in the construction of block ciphers, having the advantage that encryption and decryption operations are very similar, even identical in some cases, requiring only a reversal of the key schedule. A Feistel network is an iterated cipher with an internal function called a round function.

ch diffusion layers tend

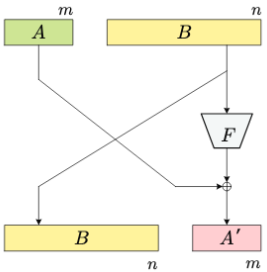
Generalized Feistel Networks (GFN) rely permutation to provide diffusion betwe also be fitted in this category. In th implementation is a bit cheaper becaus into larger groups, meaning that it is not bits one by one.



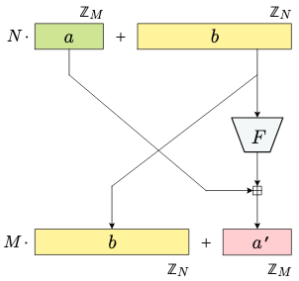
::.. Trends in LC (8/13)



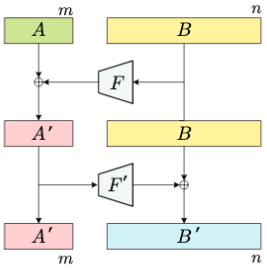
Classical Feistel
FEISTEL$^{\pi}[n]$



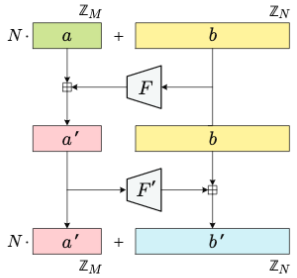
Unbalanced Feistel
Feistel$^{\pi}[m, n]$



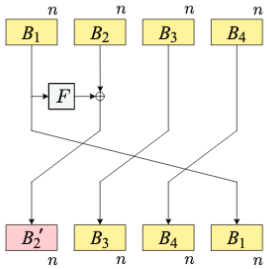
Unbalanced Numeric Feistel
Feistel$^{\pi}_{\#}[M, N]$



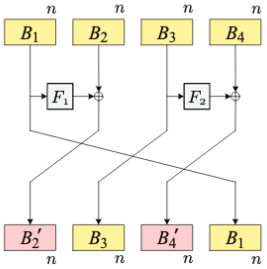
Alternating Feistel
FeIsTeL$^{\pi}[m, n]$



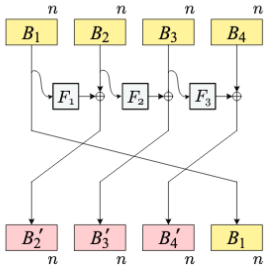
Alternating Numeric Feistel
FeIsTeL$^{\pi}_{\#}[M, N]$



Type-1 Feistel
Feistel1$^{\pi}[k, n]$



Type-2 Feistel
Feistel2$^{\pi}[k, n]$



Type-3 Feistel
Feistel3$^{\pi}[k, n]$

::.. Trends in LC (9/13)

On the other hand, diffusion can be fairly slow in the case of standard GFN, meaning that many rounds are necessary. To mitigate this problem, nibble permutations offering faster diffusion than the usual rotation can be used.

To find a compromise between diffusion, cost in software, and cost in hardware, a popular choice is to build a linear layer using word-wise rotations along with word-wise XORs.

Such a method has the advantage of allowing an efficient software implementation while remaining cheap in hardware since the rotations correspond to simple bit permutations.

::.. Trends in LC (10/13)

The key schedule is the area where lightweight algorithms differ the most from their non lightweight counterparts.

- For algorithms intended to run on standard computers, it is fine to have a complex key schedule as it would typically be run only once, the corresponding subkeys being subsequently stored.
- For lightweight algorithms, the incurred cost in terms of RAM or gate area is unacceptable. Furthermore, several lightweight algorithms dismiss resilience against related key attacks altogether, a design decision which authorizes the use of much simpler key schedules.

::.. Trends in LC (11/13)

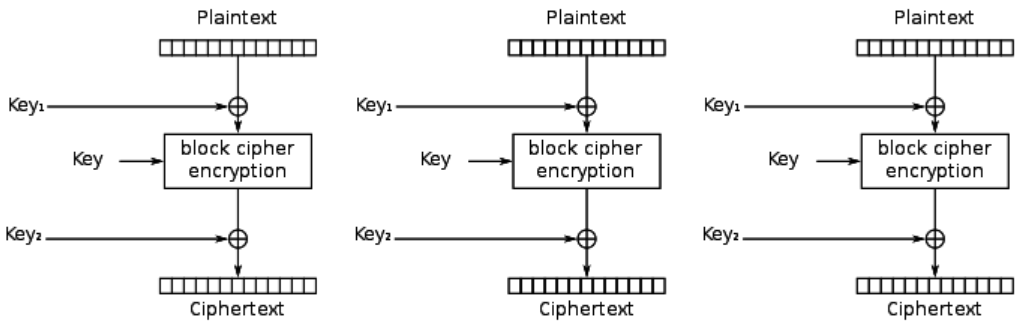
A related-key attack is any form of cryptanalysis where the attacker can observe the operation of a cipher under several different keys whose values are initially unknown, but where some mathematical relationship connecting the keys is known to the attacker.

Some cipher designers claim resilience against related-key attacks while some other algorithms are trivially vulnerable against such attacks.

Preventing related-key attacks has a cost since it implies the use of more rounds and/or more complex key schedules which lead to a performance degradation particularly unwelcome in the lightweight setting.

::.. Trends in LC (11/13)

Lightweight algorithms use a key schedule which merely selects different bits of the master key in each round for use as subkey material along with some round constants. If the master key of a block cipher is simply XORed to the internal state during each round along with a round constant, the key schedule is a variant of the Even-Mansour construction.



Xor Encrypt Xor (XEX) mode encryption

The "two-key Even–Mansour scheme": XOR the plaintext with a pre-whitening key, apply a publicly known unkeyed permutation to the result, and then XOR a post-whitening key to the permuted result.

::.. Trends in LC (13/13)

It is also possible to use different chunks of the master key during encryption. This is a selecting key schedule since it merely selects some bits of the master key for use as subkeys.

The main advantage of such methods is that they require very little logic to compute the round keys. Furthermore, they have no need for a key state getting updated at each round which would be particularly expensive in hardware.

A simple strategy to have a substantial key schedule while minimising its cost is to reuse significant parts of the round function to update the key state. The whole round function can be used, or only parts of it.

::.. Trade-Offs in LC (1/3)

Good performances and good security are at odds against one another. While this trade-off is not specific to lightweight cryptography, it is more pressing in this context because of the greater performance constraints that occur in this case.

- The simplest is the number of rounds. Using fewer leads to a lower latency and, depending on the implementation, to a higher throughput. However, it also leads to a decreased security margin and may in fact lead to cryptanalysis.
- Overall, choosing the right number of rounds for a primitive requires a difficult balance between speed and security.

Using a strong S-Box with a high algebraic degree, low differential uniformity and low linearity would increase the resilience of the algorithm against common attacks. However, such components are more expensive to implement.

::.. Trade-Offs in LC (2/3)

A trade-off more specific to lightweight cryptography is that of specialization versus versatility.

- Some solutions have been optimized for a low gate count in hardware implementation. On the other hand, the huge number of rounds mean that a low latency implementation would be very difficult to design.
- Low latency block ciphers uses a more complex round function which uses different binary matrices for different parts of the internal state, which would complicate a serialized implementation.
- Some algorithms have been explicitly designed to provide good performances across many different platforms by relying only on word-wide logical operations and rotations.

::.. Trade-Offs in LC (3/3)

Algorithms for which a lower security level was deemed sufficient by the designers tend to be designed with a specific platform and use case in mind. On the other hand, more generalist algorithms make comparatively more conservative choices.

We thus see two subsets of algorithms, some being very specialized and willingly making bolder design choices to improve performances while others strive for versatility and more conservative security margins.

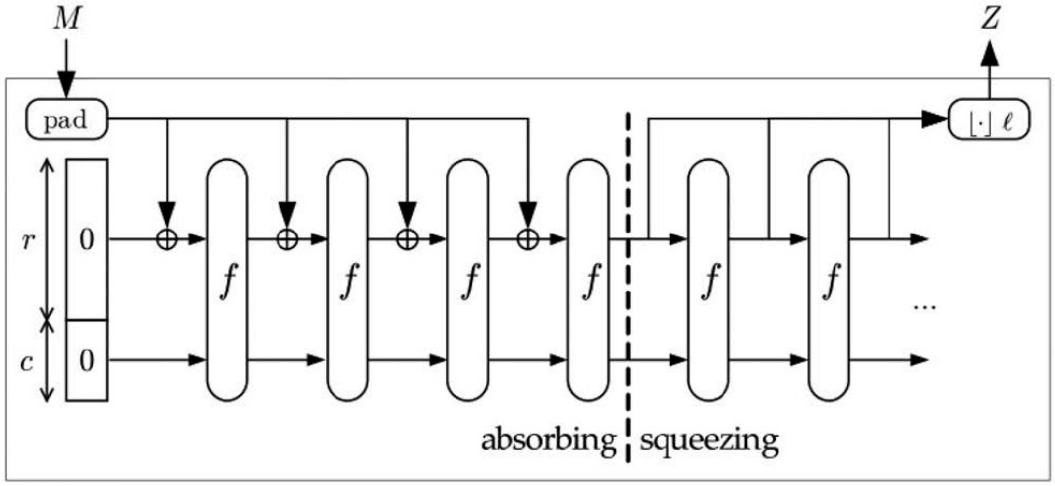
::.. Hashing (1/6)

While we will all have 32-bit or 64-bit processors in our mobile phones and desktops and have much more the 1GB of memory, in an IoT world we often measure memory capacity in just a few kilobytes and where 8-bit processors rule the roost.

So, crypto hash functions for MD5 and SHA-1, and most of our other modern hash methods, are just not efficient for IoT devices. NIST have thus recommended new hashing methods able produce a much smaller memory footprint and have a target an input of just 256 characters (whereas typically hash functions support up to 264 bits).

::.. Hashing (2/6)

SPONGENT uses the sponge function, where there is a fixed-length permutation (or transformation) and a padding rule, that transforms the input into blocks of r bits. This construction thus takes a variable-length input and maps it to a variable-length output. Overall, the method uses a finite-state machine process and iterates through the states with the addition of the input data by using either a publicly known unkeyed permutation (P-Sponge) or with a random function (T-Sponge).

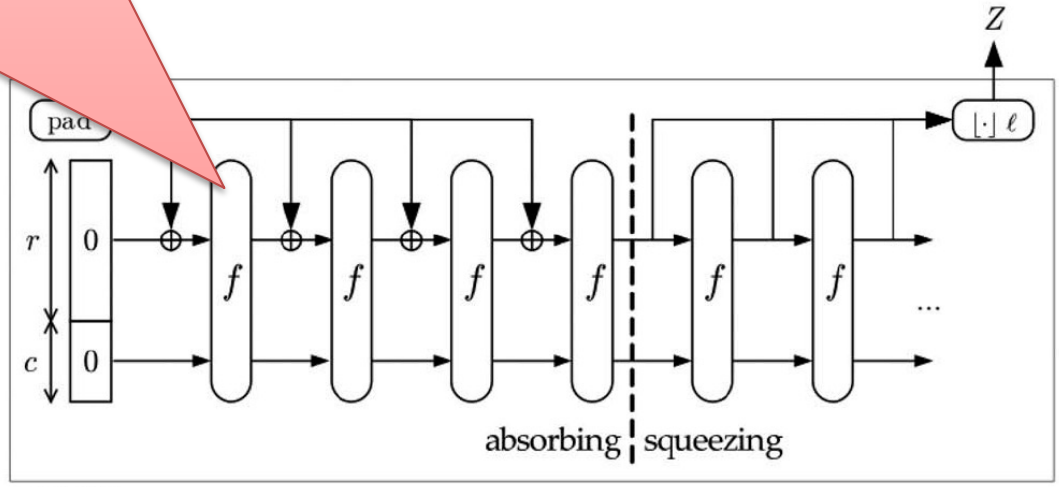


sponge

::.. Hashing (2/6)

The sponge construction uses a function f which has a variable-length input and a defined output length. It operates on a fixed number of bits (b) – the width. The sponge construction then operates on a state of b = r + c bits. r is defined as the bitrate and c as the capacity.

…here there is a fixed-…nd a padding rule, that … This construction thus …it to a variable-length …-state machine process … addition of the input …keyed permutation (P-…nge).



sponge

::.. Hashing (2/6)

SPONGENT uses the sponge function, where there is a fixed-length permutation (or transformation) and a padding rule, that transforms the input into blocks of r bits. This construction thus

Initially an input string is padded using a reversible padding rule (such as adding NULL characters) and then segmented into blocks of r bits. Next, the b bits of the state

it to a variable-length

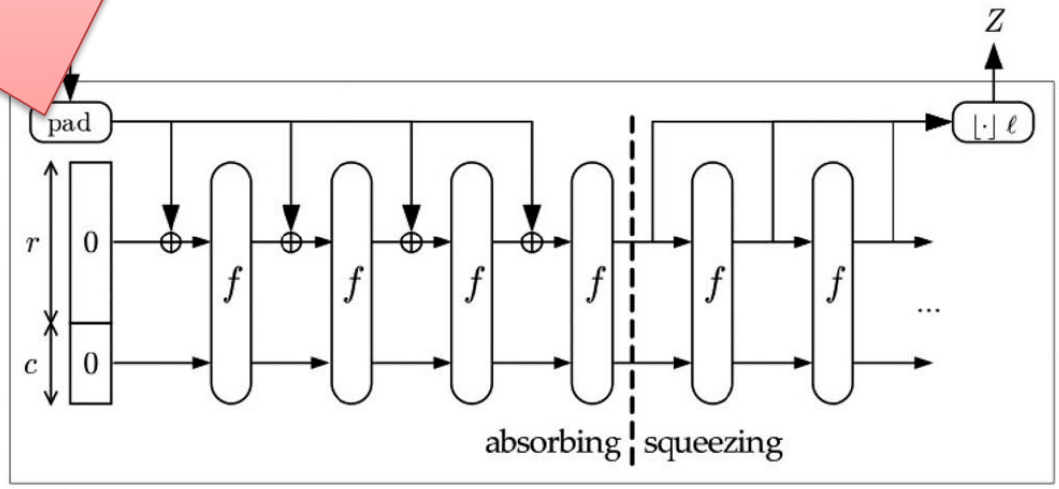
-state machine process

addition of the input

keyed permutation (P-

are set to zero.

nge).



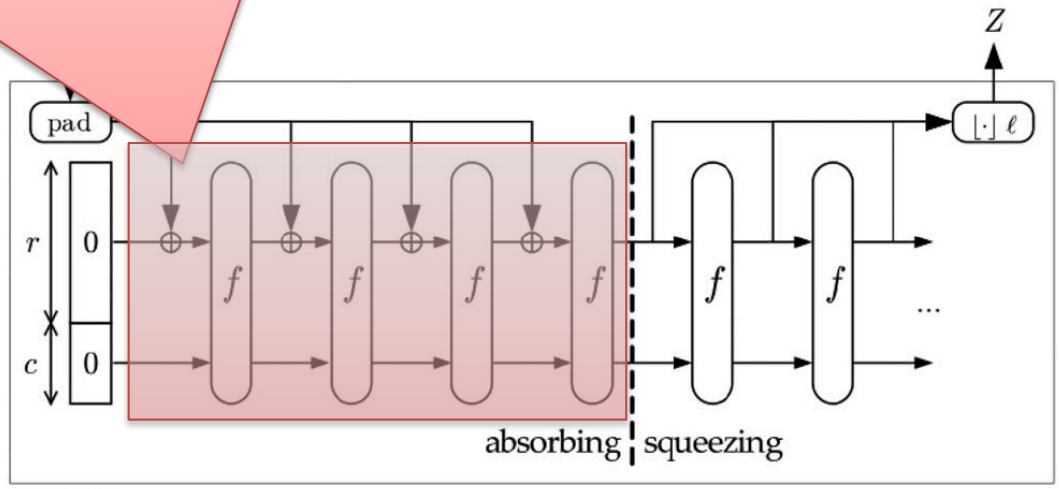
sponge

::.. Hashing (2/6)

SPONGENT uses the sponge function, where there is a fixed-length permutation (or transformation) and a padding rule, that transforms the input into blocks of r bits. This construction thus

Absorbing phase. This is where the r-bit input blocks are X-ORed into the first r bits of the state, interleaved with applications of the function f. After all the input blocks have been processed, we then move to a squeezing phase.

a variable-length machine process tion of the input l permutation (P-

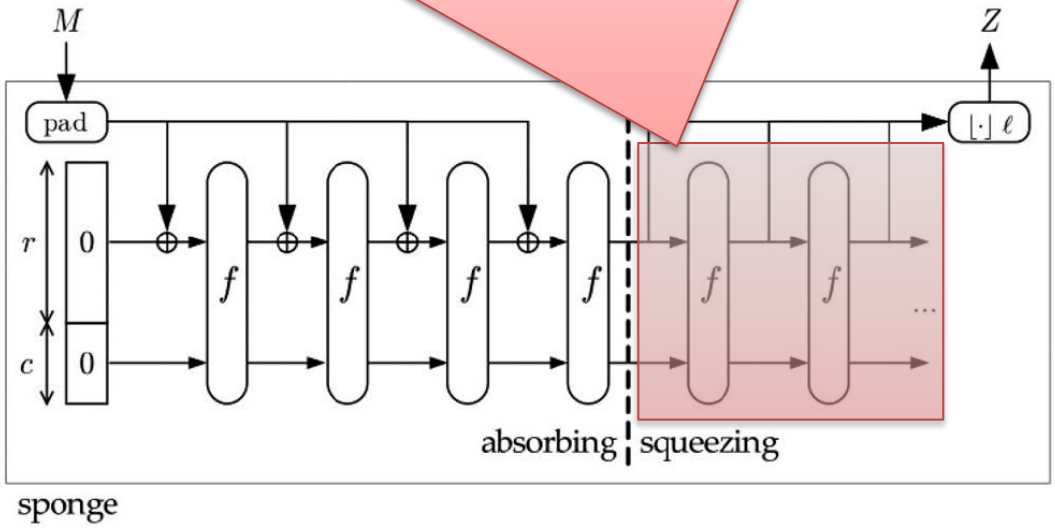


sponge

::.. Hashing (2/6)

SPONGENT uses the sponge function, where there is a fixed-length permutation (or transformation) and a padding rule, that transforms the input into blocks of r bits. This construction thus takes a variable-length input and maps it to a variable-length output. Overall
and iterates th
data by using
Sponge) or wit

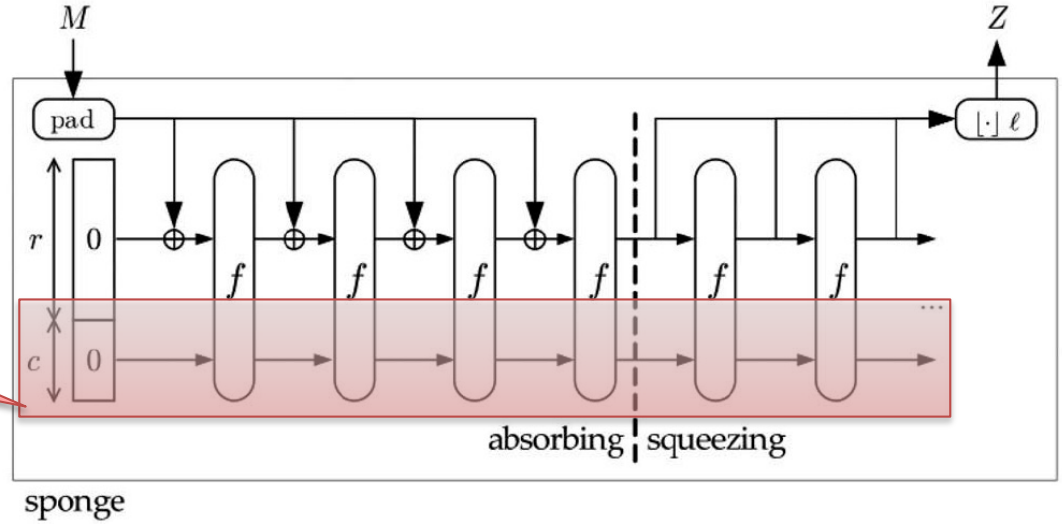
Squeezing phase. This is where the first r bits of the state are outputted as blocks and interleaved with the function f. The number of bits of the output is defined as part of the process.



::.. Hashing (2/6)

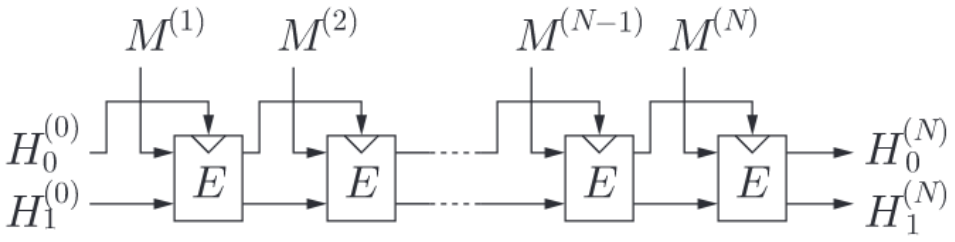
SPONGENT uses the sponge function, where there is a fixed-length permutation (or transformation) and a padding rule, that transforms the input into blocks of r bits. This construction thus takes a variable-length input and maps it to a variable-length output. Overall, the method uses a finite-state machine process and iterates through the states with the addition of the input data by using either a publicly known unkeyed permutation (P-Sponge) or with a random function (T-Sponge).

Overall the last c bits of a state are never changed by the input blocks and never output within the squeezing phase.



::.. Hashing (3/6)

Lesamnta-LW uses an AES-based design so as to gain clear advantages over known block-cipher based designs such as SHA-256 and MAME. The key scheduling function ensures a strong non-linearity and an excellent diffusion property by re-using the 32-bit permutation of the mixing function; this reduces the hardware complexity since a part of the hardware can be reused.

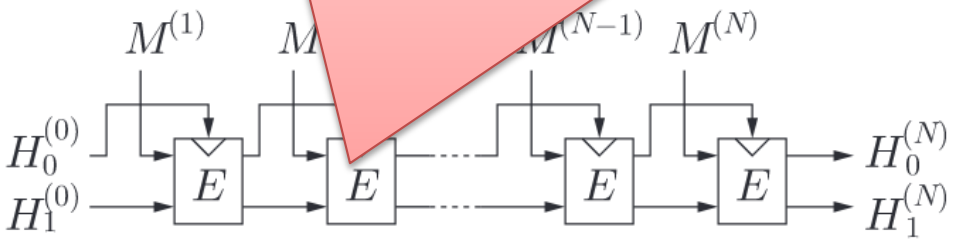


It is a Merkle-Damgård iterated hash function, i.e., a method of building collision-resistant cryptographic hash functions from collision-resistant one-way compression functions.

::.. Hashing (3/6)

Lesamnta-LW uses an AES-based design so as to gain clear advantages over known block-cipher based designs such as SHA-256 and MAME. The key scheduling function ensures a

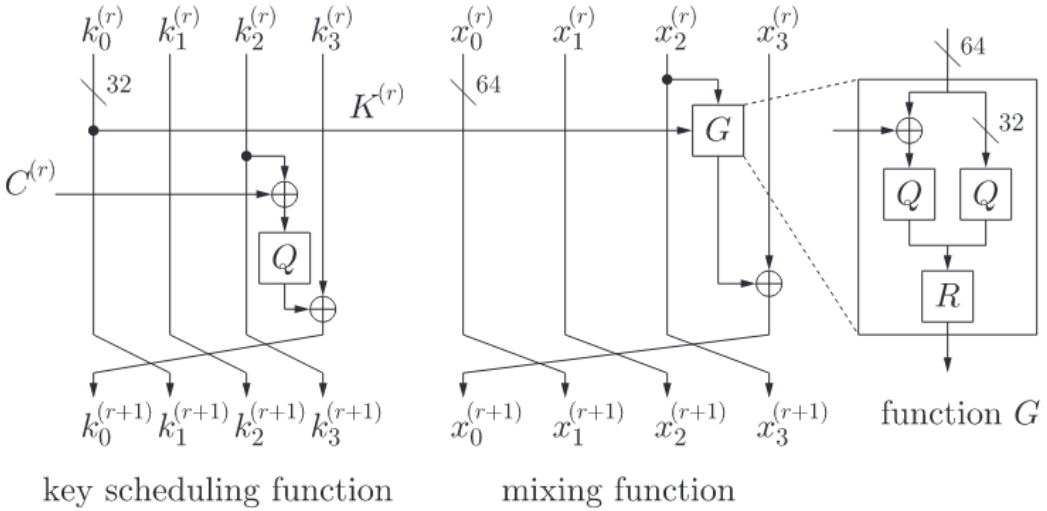
In the M-D construction, a one-way compression function is denoted by f, and transforms two fixed length inputs to an output of the same size as one of the inputs. Here E, is a a 256-bit AES block cipher with a 128-bit key K, acting as one-way compression function.



It is a Merkle-Damgård iterated hash function, i.e., a method of building collision-resistant cryptographic hash functions from collision-resistant one-way compression functions.

::.. Hashing (4/6)

Lesamnta-LW uses a 64-round block cipher E that takes as input a 128-bit key and a 256-bit plaintext. The block cipher consists of two parts: the key scheduling function mapping the key to the round keys and the mixing function taking as input a plaintext and the round keys to produce a ciphertext.

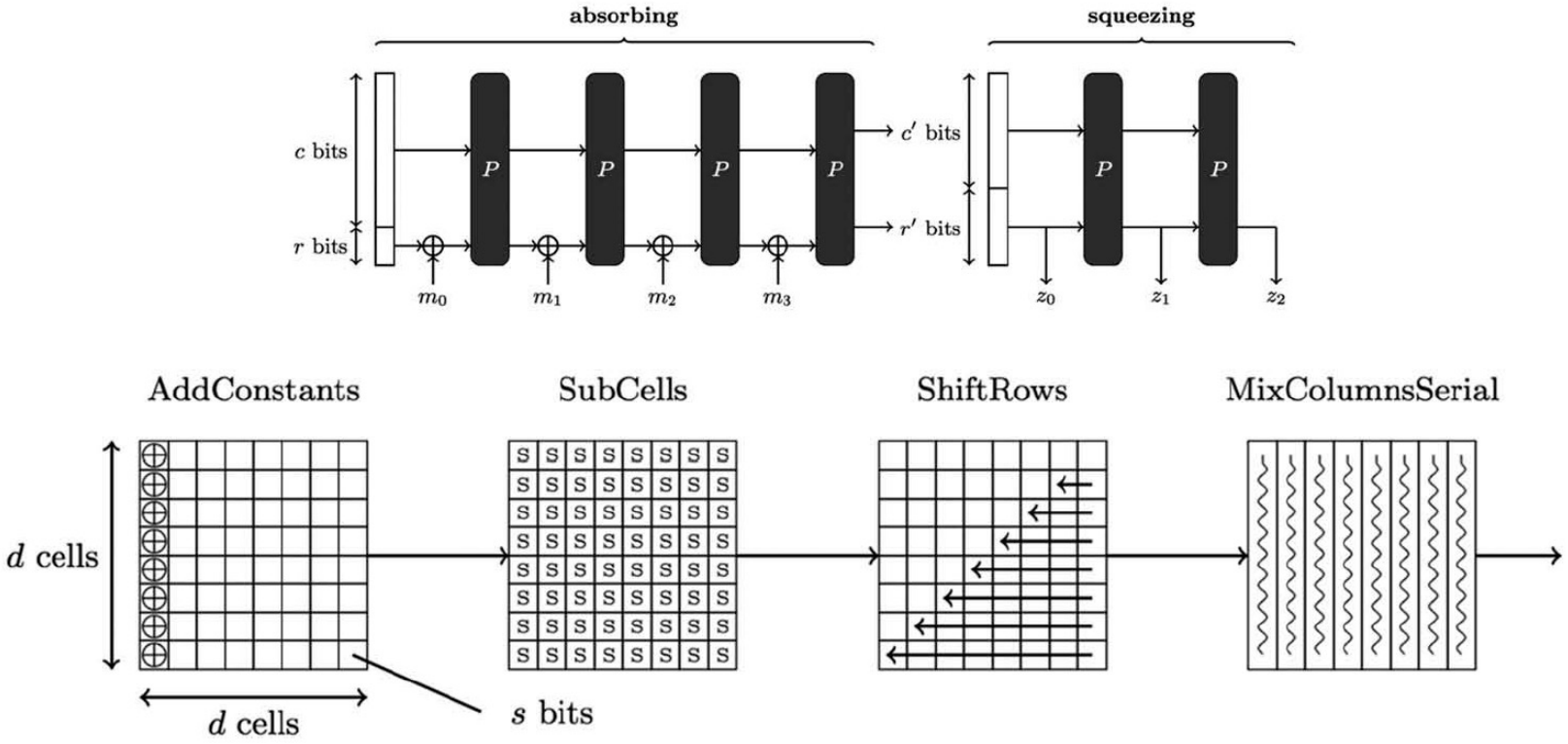


Both of them use a type-1 4-branch generalized Feistel net-work (GFN). The mixing function consists of XORs, a word wise permutation, and a non-linear function G.

G consists of XOR operations, a 32-bit non-linear permutation Q (i.e., a Sub Bytes transformation with AES S-box and AES MDS matrix multiplication), and a function R making permutation.

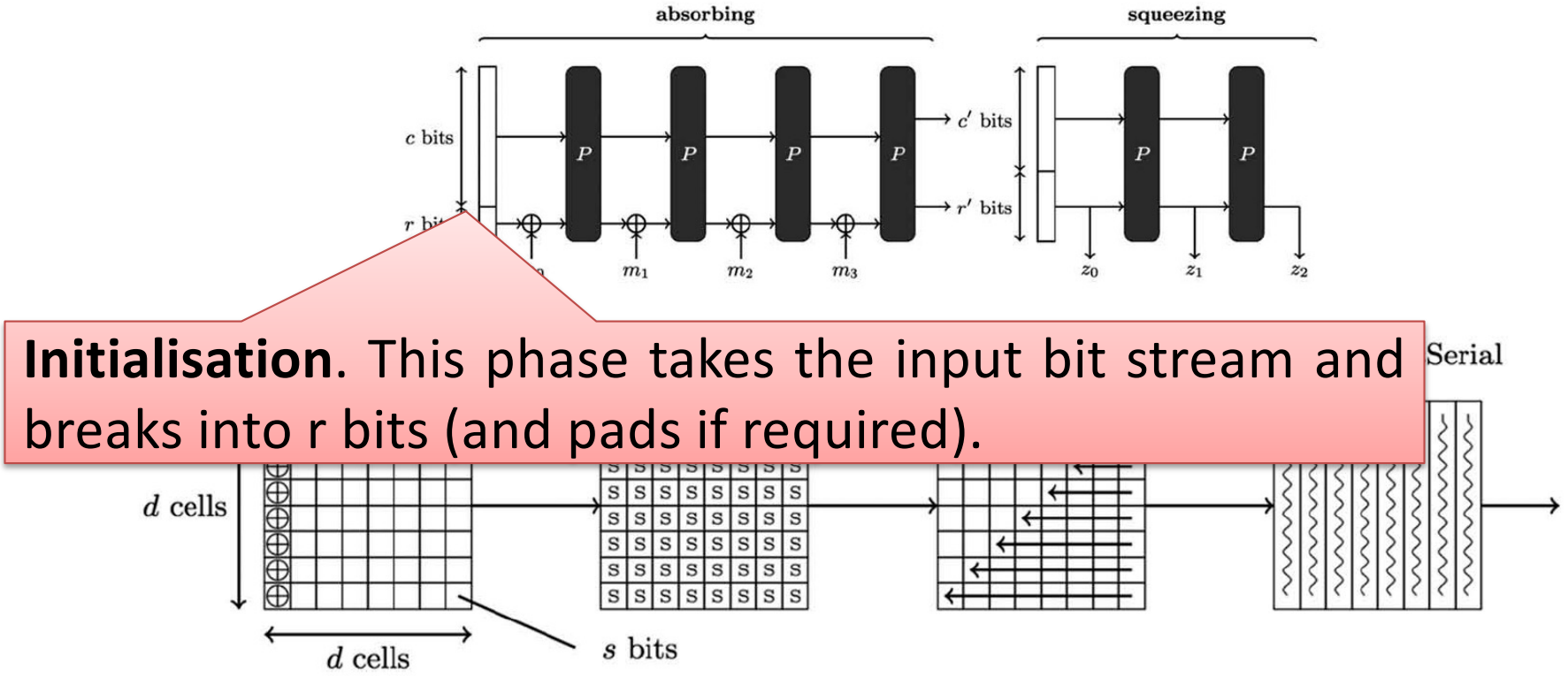
::.. Hashing (5/6)

PHOTON is lightweight cryptography method for hashing and is based on an AES type approach, using a sponge function and where we take input bits and XOR with bits taken from the current state. Overall, there are three main phases:



::.. Hashing (5/6)

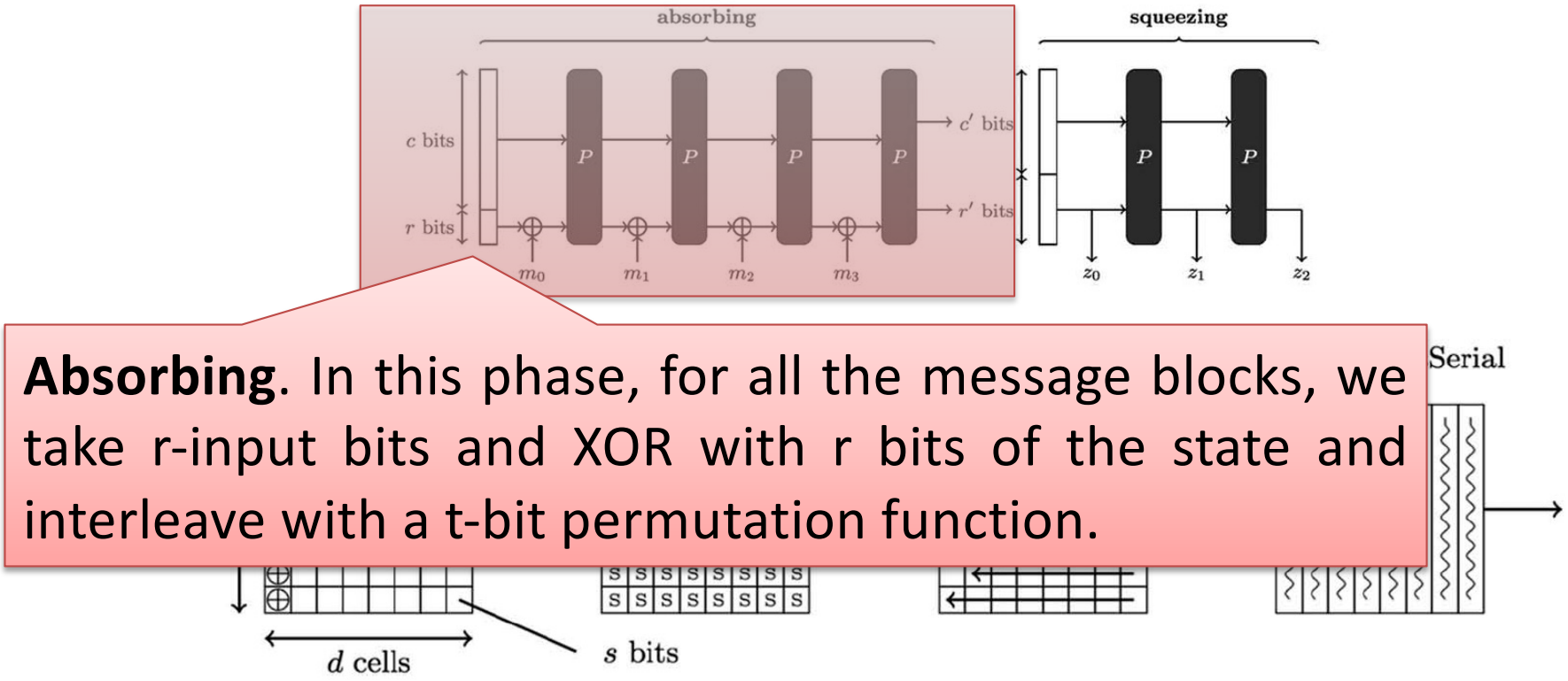
PHOTON is lightweight cryptography method for hashing and is based on an AES type approach, using a sponge function and where we take input bits and XOR with bits taken from the current state. Overall, there are three main phases:



Initialisation. This phase takes the input bit stream and breaks into r bits (and pads if required).

::.. Hashing (5/6)

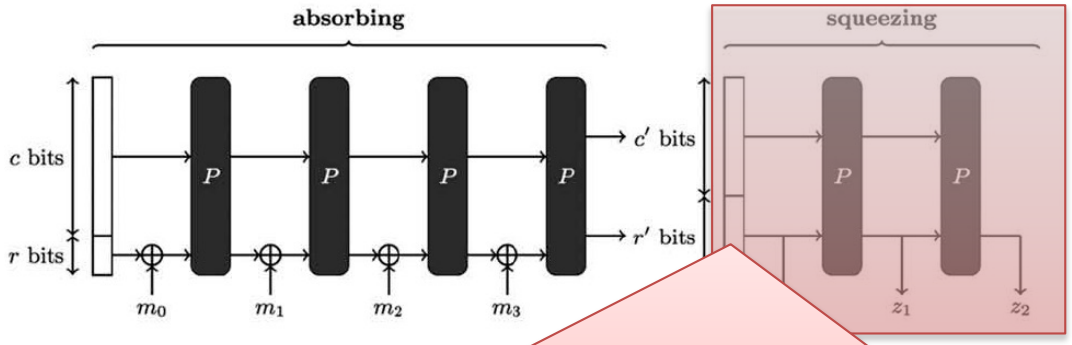
PHOTON is lightweight cryptography method for hashing and is based on an AES type approach, using a sponge function and where we take input bits and XOR with bits taken from the current state. Overall, there are three main phases:



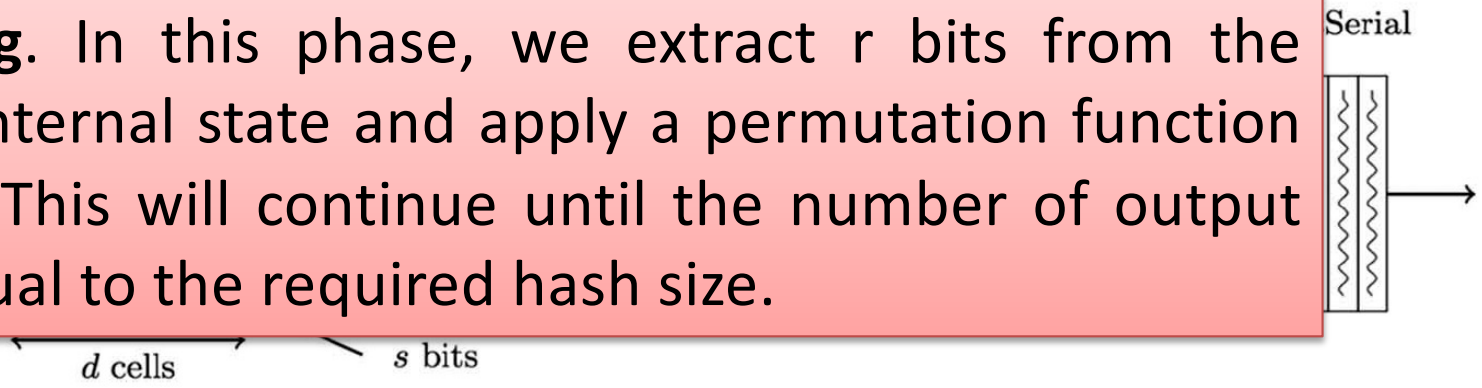
Absorbing. In this phase, for all the message blocks, we take r-input bits and XOR with r bits of the state and interleave with a t-bit permutation function.

::.. Hashing (5/6)

PHOTON is lightweight cryptography method for hashing and is based on an AES type approach, using a sponge function and where we take input bits and XOR with bits taken from the current state. Overall, there are three main phases:



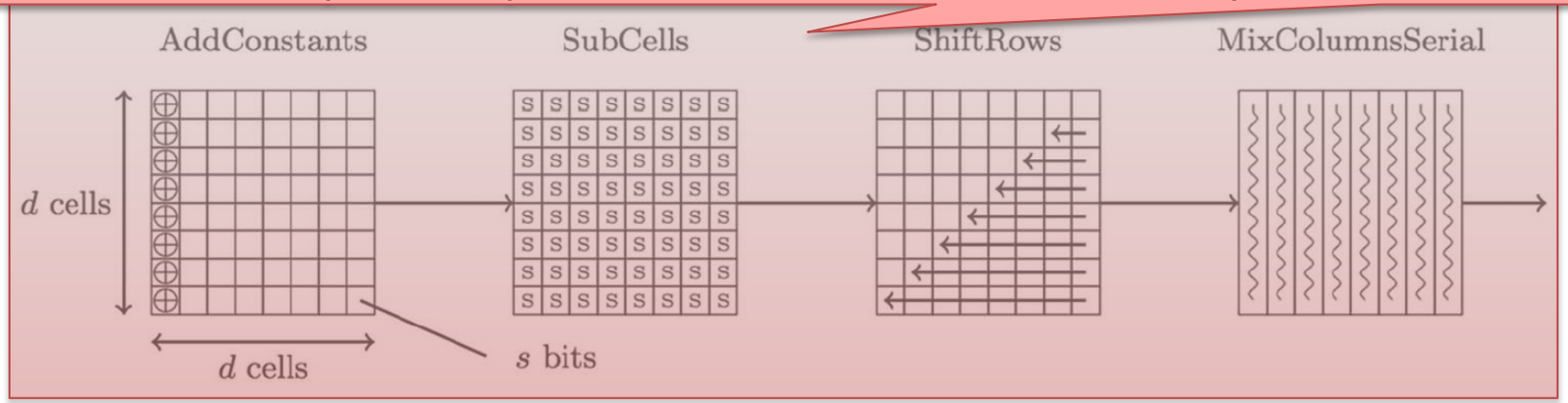
Squeezing. In this phase, we extract r bits from the current internal state and apply a permutation function (P) to it. This will continue until the number of output bits is equal to the required hash size.



::.. Hashing (5/6)

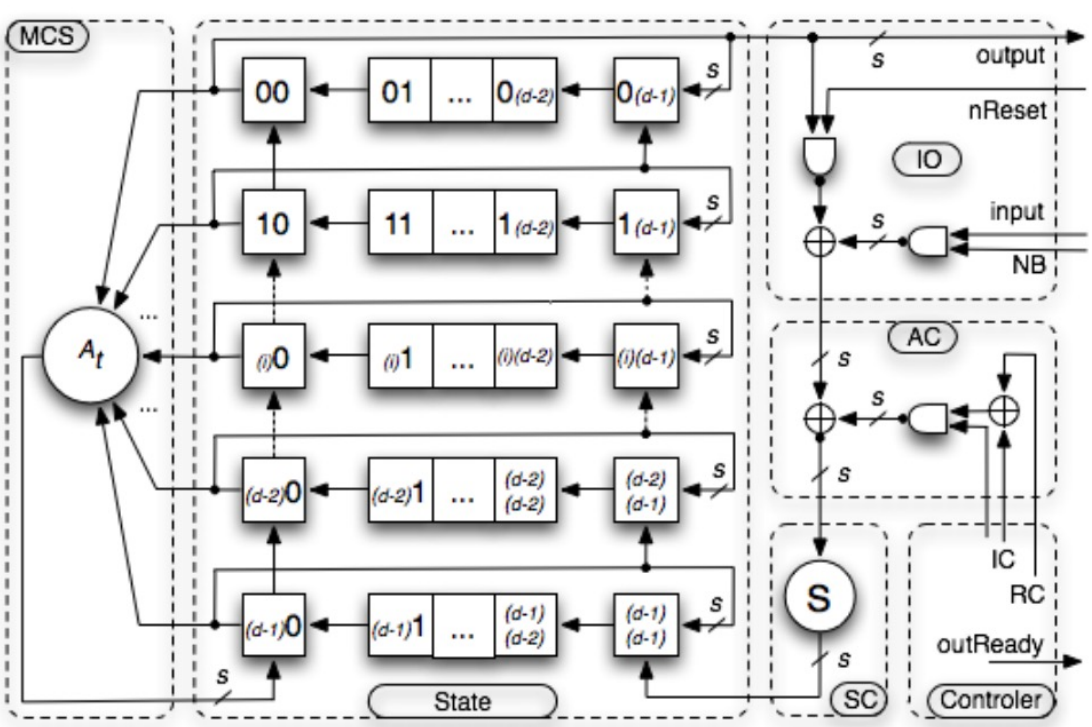
The internal permutation function (P) is similar to AES with 12 rounds and which each round has the functions of

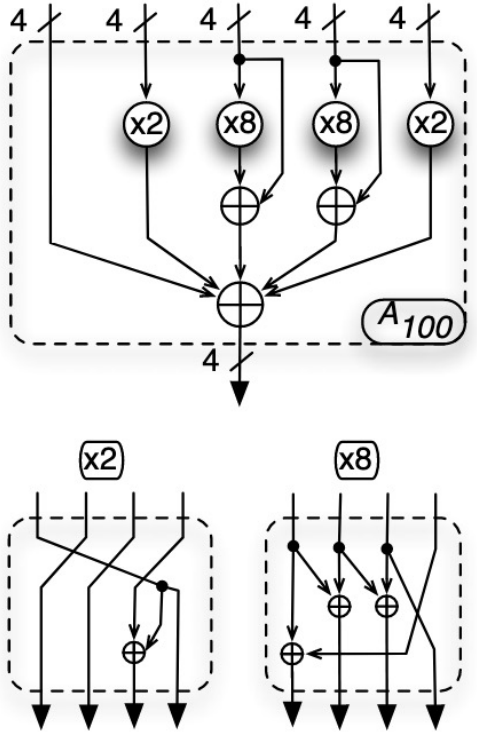
- AddConstants. In this function, the first column with the internal state is XOR-ed with round (r) and internal constants.
- SubCells. In this function, the internal state is fed through the PRESENT S-box .
- ShiftRows. In this function, the internal state cell row [i] is cyclically shifted by i positions to the left.
- MixColumnsSerial. In this function, the internal state cell column is multiplied by the Maximum Distance Separable matrix.



::.. Hashing (6/6)

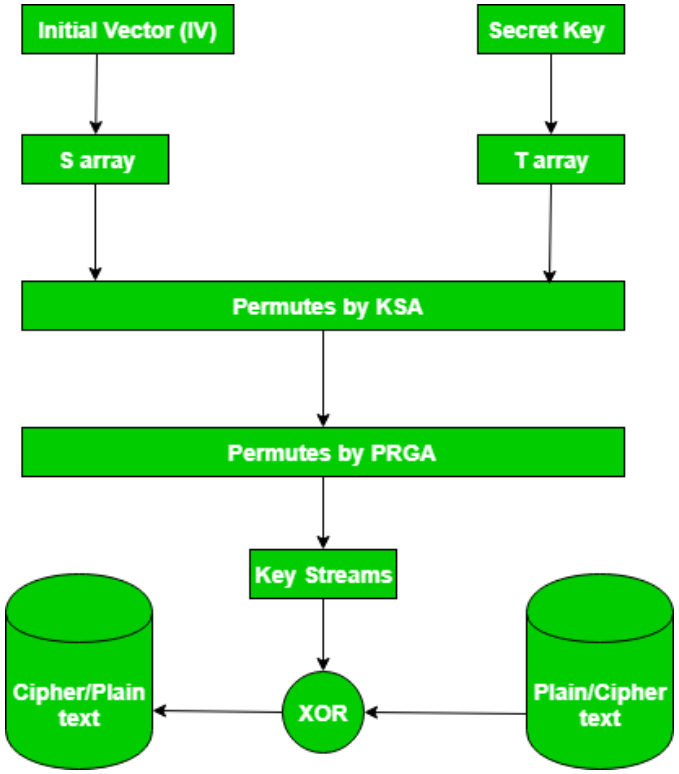
Serial hardware architecture of PHOTON performs operations on one cell per clock cycle, and aims for the smallest area possible; the second one is a d times parallelization of the first architecture, thus performing operations on one row in one clock cycle, resulting in a significant speed-up.





::.. Streaming Ciphers (1/6)

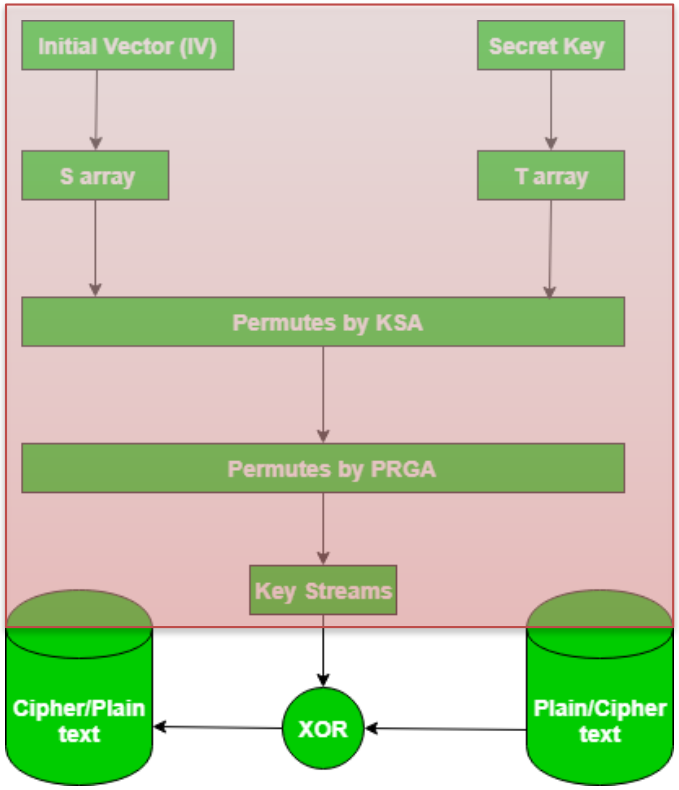
RC4 is a stream cipher and variable length key algorithm, encrypting one byte at a time (or larger units on a time).



::.. Streaming Ciphers (1/6)

RC4 is a stream cipher and variable length key algorithm, encrypting one byte at a time (or larger units on a time).

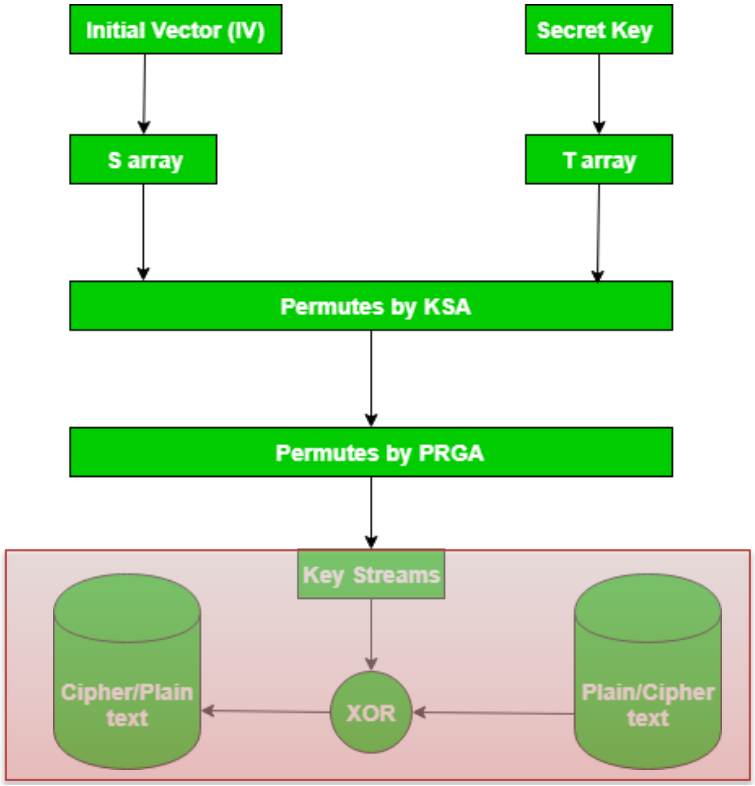
A pseudorandom bit generator produces a stream 8-bit number that is unpredictable without knowledge of input key.



::.. Streaming Ciphers (1/6)

RC4 is a stream cipher and variable length key algorithm, encrypting one byte at a time (or larger units on a time).

A pseudorandom bit generator produces a stream 8-bit number that is unpredictable without knowledge of input key.

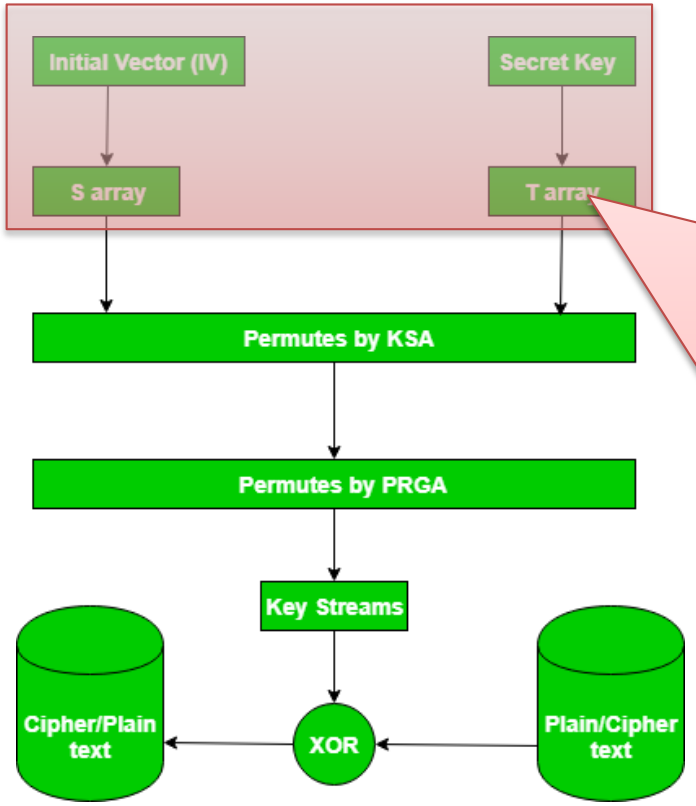


The key-stream is combined one byte at a time with the plaintext stream cipher using X-OR operation.

::.. Streaming Ciphers (1/6)

RC4 is a stream cipher and variable length key algorithm, encrypting one byte at a time (or larger units on a time).

A pseudorandom bit generator produces a stream 8-bit number that is unpredictable without knowledge of input key.



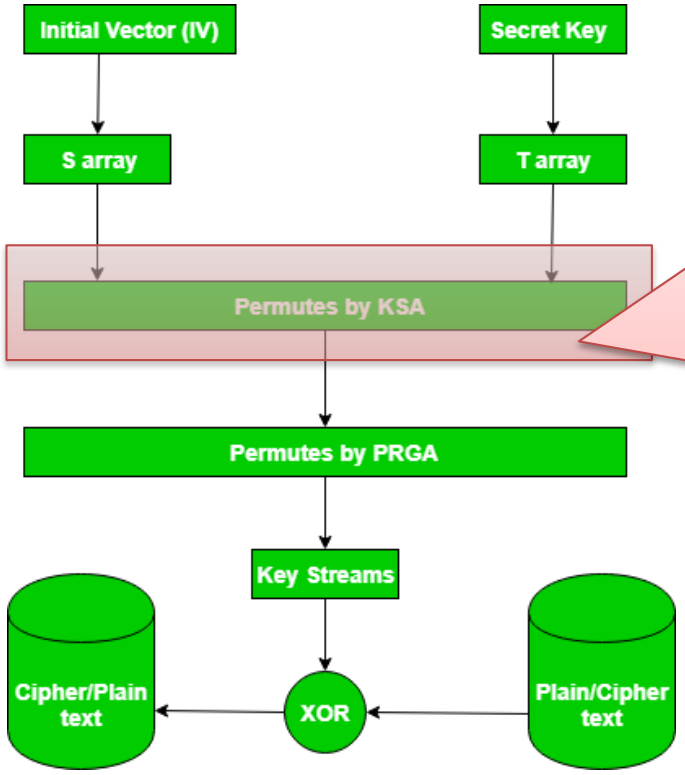
The key-stream is combined one byte at a time with the plaintext

The entries of S are set equal to the values from 0 to 255, a temporary vector T is created equal to the provided key if its length is 256 bytes. Otherwise, for a key with a smaller length k-len, the first k-len elements of T as copied from K and this is repeated as many times as necessary to fill T.

::.. Streaming Ciphers (1/6)

RC4 is a stream cipher and variable length key algorithm, encrypting one byte at a time (or larger units on a time).

A pseudorandom bit generator produces a stream 8-bit number that is unpredictable without knowledge of input key.



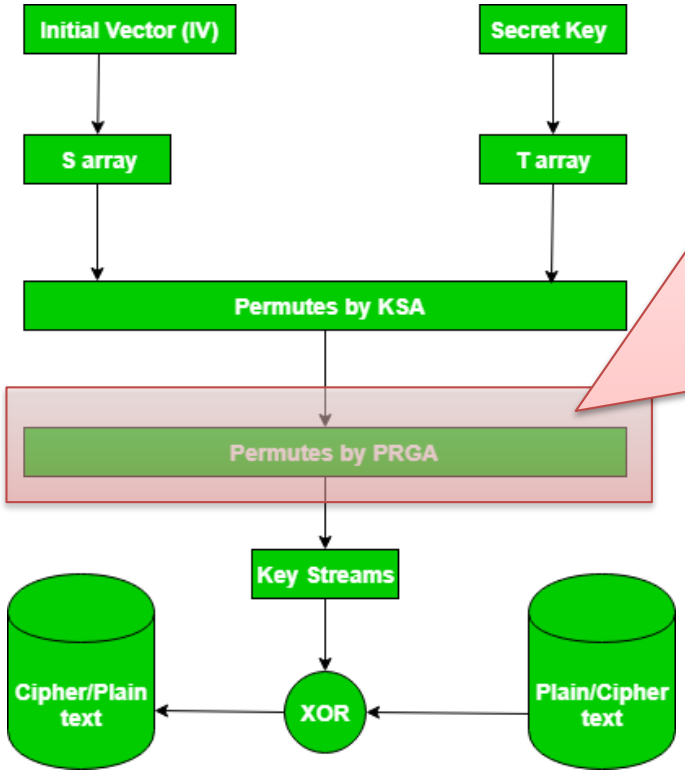
The key-stream is combined one byte at a time with the plaintext stream cipher using X-OR operation.

Key Scheduling Algorithm (KSA): T is used to produce the initial permutation of S. For each S[i], the algorithm swaps it with another bit in S according to a scheme dictated by T[i].

::.. Streaming Ciphers (1/6)

RC4 is a stream cipher and variable length key algorithm, encrypting one byte at a time (or larger units on a time).

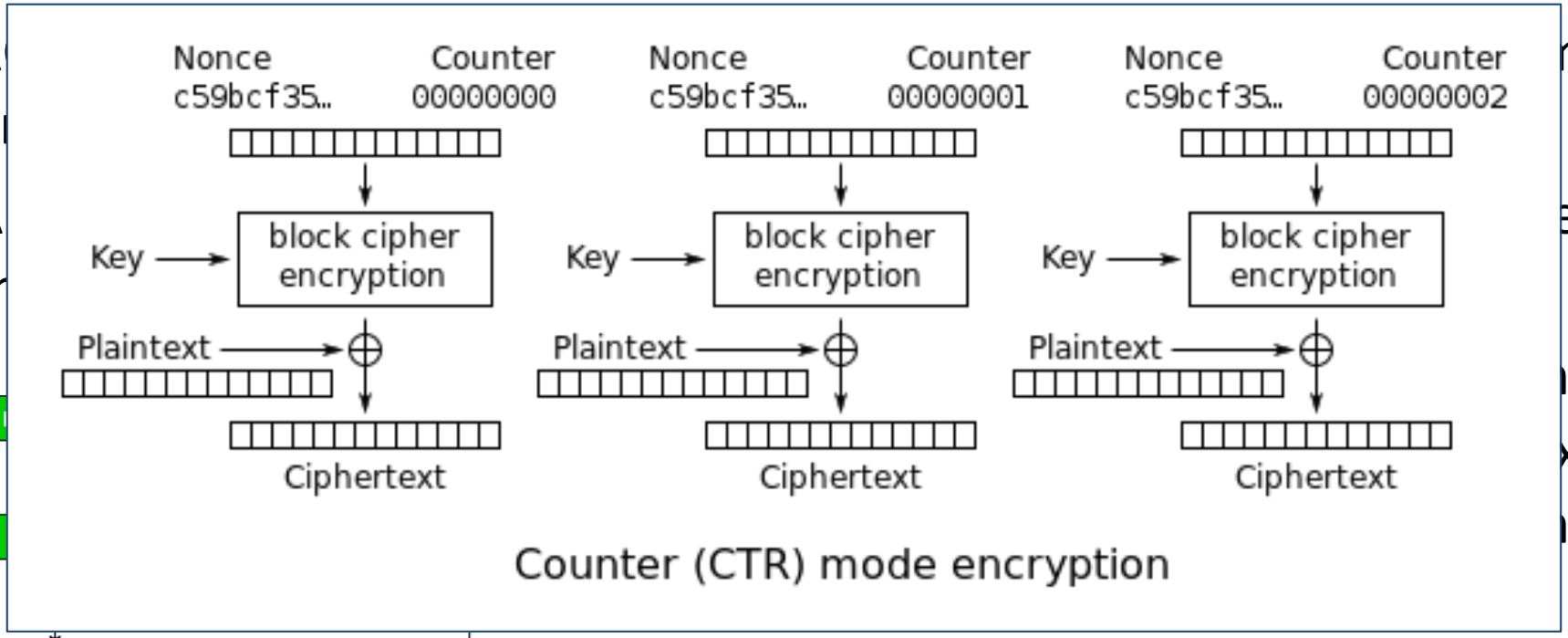
A pseudorandom bit generator produces a stream 8-bit number that is unpredictable without knowledge of input key.

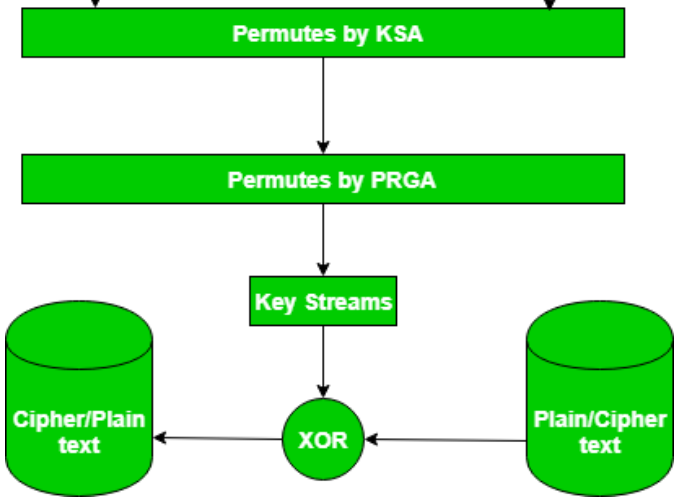


The key-stream is combined one byte at a time with the plaintext

Pseudo-Random Generation Algorithm (PRGA): for each S[i], the algorithm swaps it with another bit in S according to a scheme dictated by the current configuration of S. This is repeated multiple times, each loop generates a one-byte pseudo-random output to be xored with the plaintext.

::.. Streaming Ciphers (1/6)



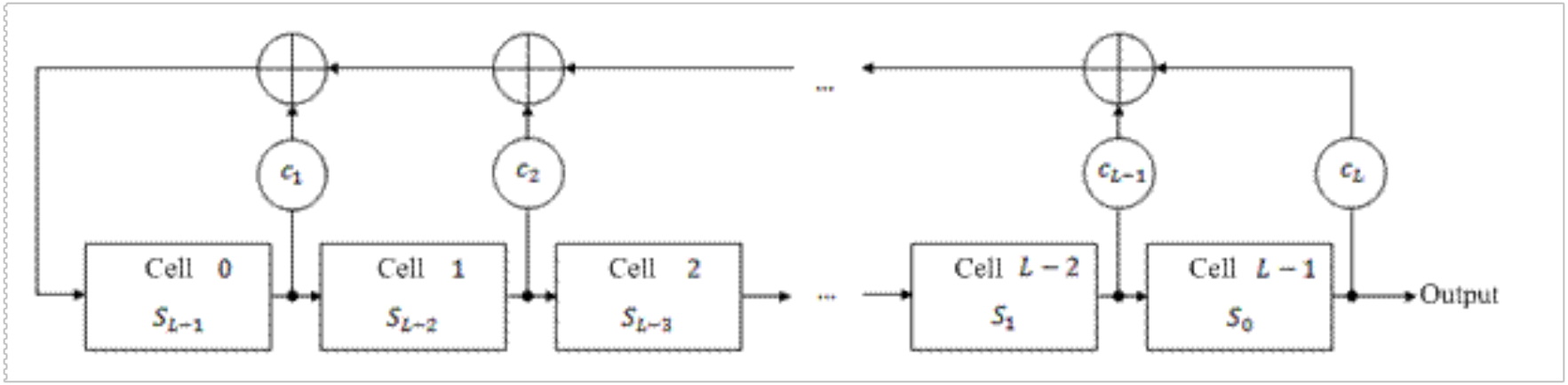


A simple way to build a stream cipher is to use a regular block cipher in counter-mode, generating the next keystream block by encrypting successive values of a "counter".

::.. Streaming Ciphers (2/6)

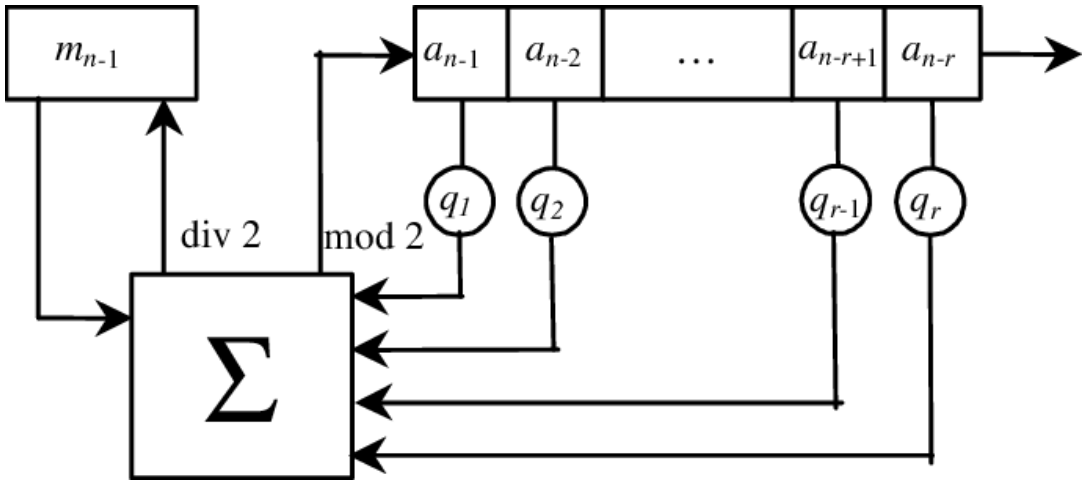
Some basic implementation mechanisms can be introduced to lower the implementation costs of RC4 or other stream ciphers:

- **Linear Feedback Shift Register (LFSR)** — a shift register of bit words, in which the input bit is a linear function (e.g., Xor) of its previous state (i.e., the content of the register cells). Both hardware and software implementations are available and used for generating pseudo-random numbers.



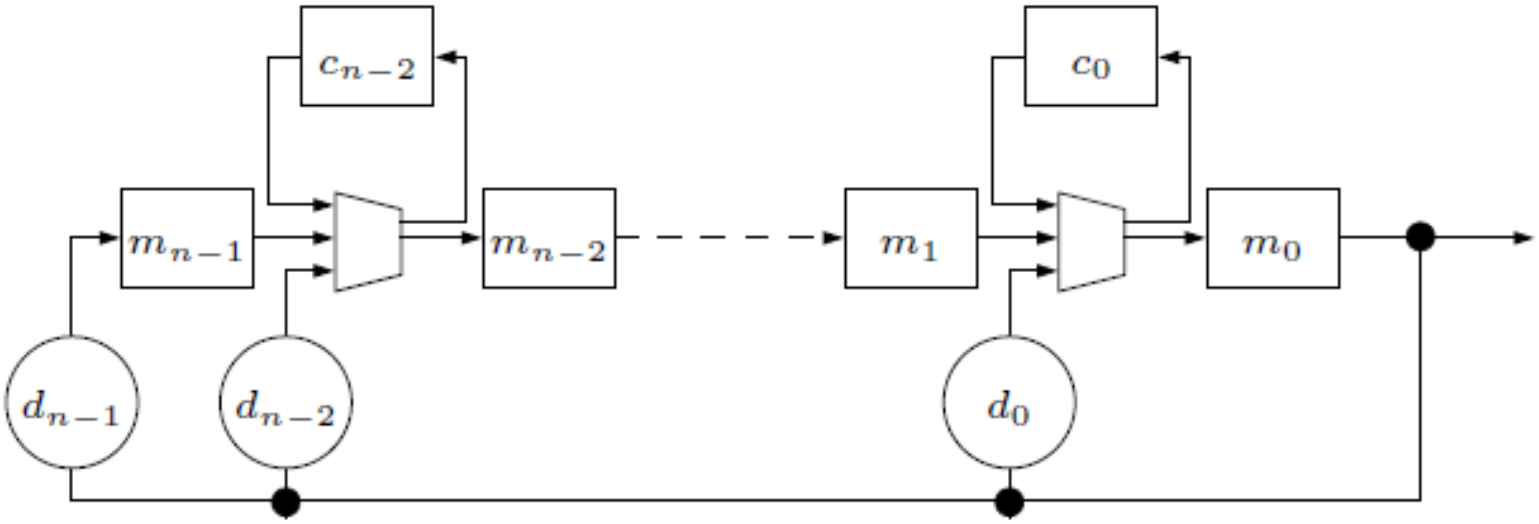
::.. Streaming Ciphers (3/6)

- **Feedback with Carry Shift Register (FCSR)** — A variation of the LFSR where a secondary register is used to store the carry. FCSR has two implementations:
  - The Fibonacci one consists in a single feedback function which depends on multiple inputs.



::.. Streaming Ciphers (3/6)

- **Feedback with Carry Shift Register (FCSR)** — A variation of the LFSR where a secondary register is used to store the carry. FCSR has two implementations:
  - The Fibonacci one consists in a single feedback function which depends on multiple inputs.
  - The Galois one has multiple feedback functions with one common input.



::.. Streaming Ciphers (3/6)

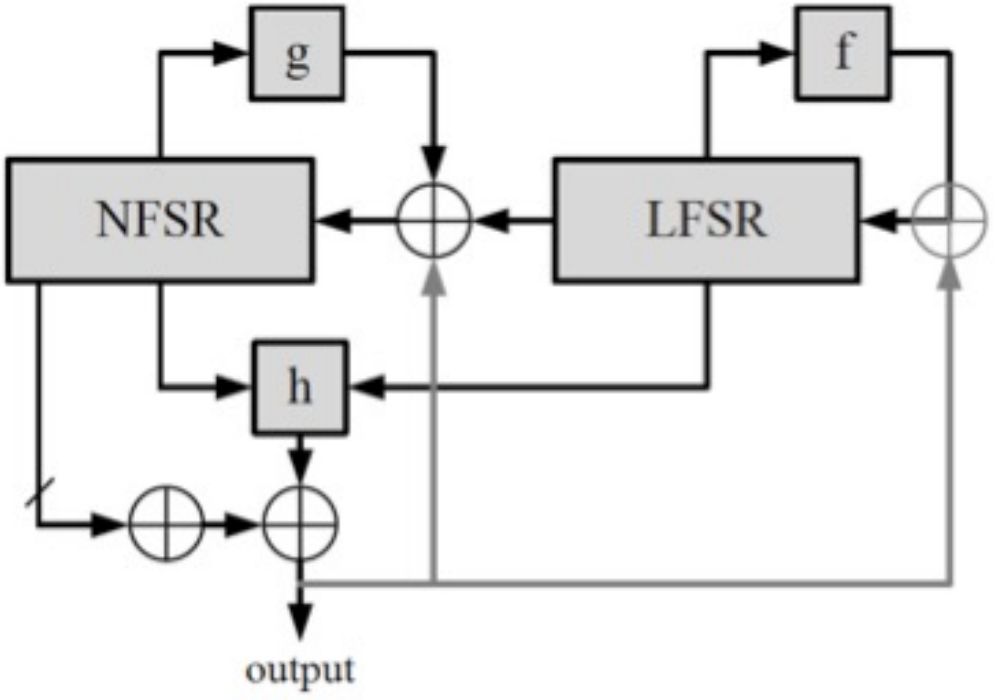
- **Feedback with Carry Shift Register (FCSR)** — A variation of the LFSR where a secondary register is used to store the carry. FCSR has two implementations:
  - The Fibonacci one consists in a single feedback function which depends on multiple inputs.
  - The Galois one has multiple feedback functions with one common input.

The F-FCSR Galois configuration is used as it is more effective; however, it is possible to have a ring FCSR viewed as a trade-off between the two previous representations.

A **Nonlinear Feedback Shift Register (NLFSR)** is a shift register whose input bit is a non-linear function of its previous state. It is more resistant to cryptanalytic attacks LFSRs.

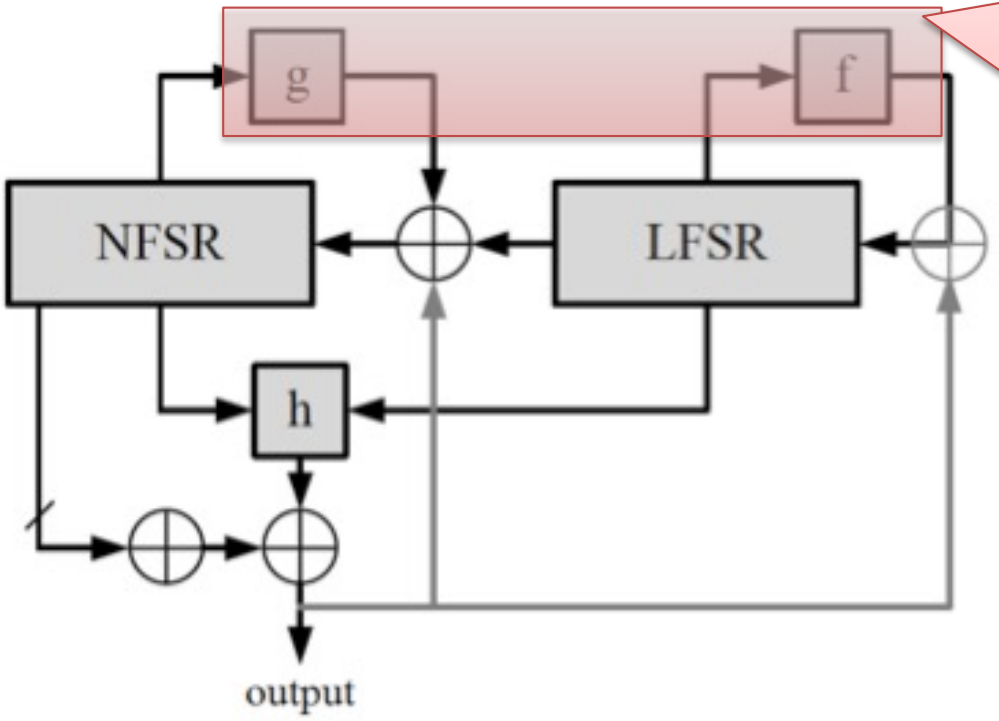
::.. Streaming Ciphers (4/6)

Grain is a bit-oriented synchronous stream cipher, whose design is very easy to implement in hardware and requires only small chip area. It consists of two 80-bit shift registers where one has a linear feedback (LFSR) and the other a nonlinear feedback (NFSR).



::.. Streaming Ciphers (4/6)

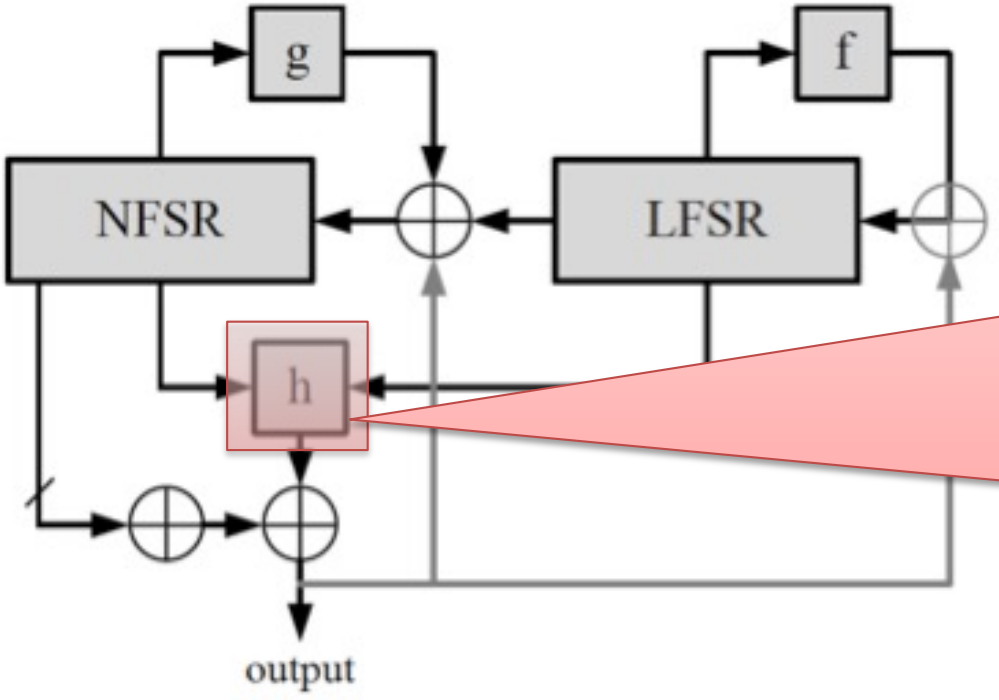
Grain is a bit-oriented synchronous stream cipher, whose design is very easy to implement in hardware and requires only small chip area. It consists of two 80-bit shift registers where one has a linear feedback (LFSR) and the other a nonlinear feedback (NFSR).



Two polynomials of degree 80, f(x) and g(x), are used as feedback function for LFSR and NFSR.

::.. Streaming Ciphers (4/6)

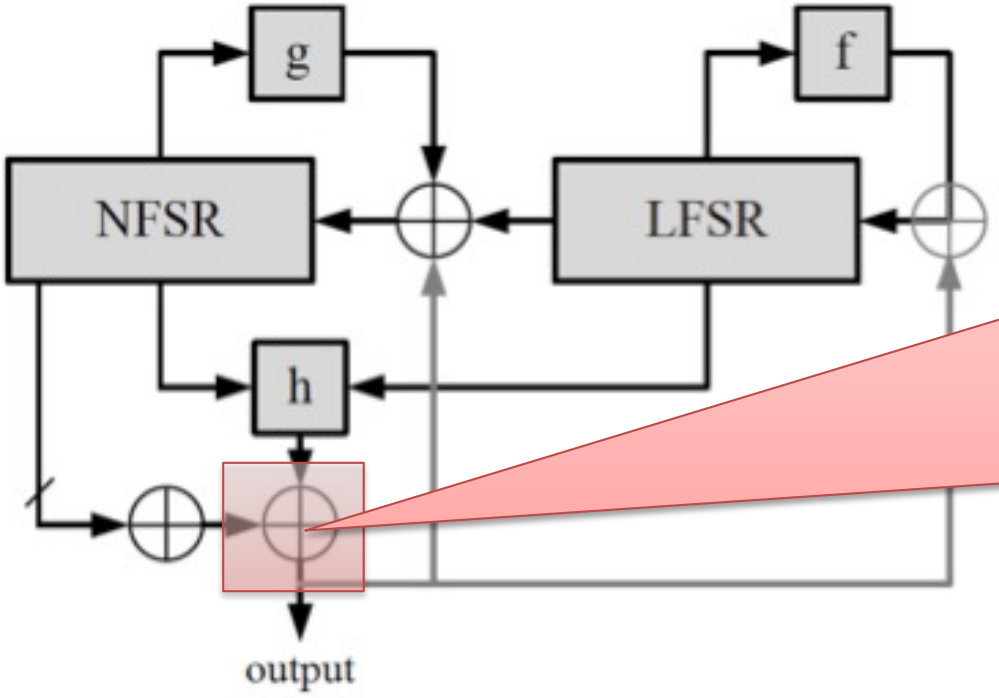
Grain is a bit-oriented synchronous stream cipher, whose design is very easy to implement in hardware and requires only small chip area. It consists of two 80-bit shift registers where one has a linear feedback (LFSR) and the other a nonlinear feedback (NFSR).



The output function h(x) uses as input selected bits from both feedback shift registers, acting as a filter function. Additionally, seven bits of NFSR are XORed together and the result is added to the function h(x).

::.. Streaming Ciphers (4/6)

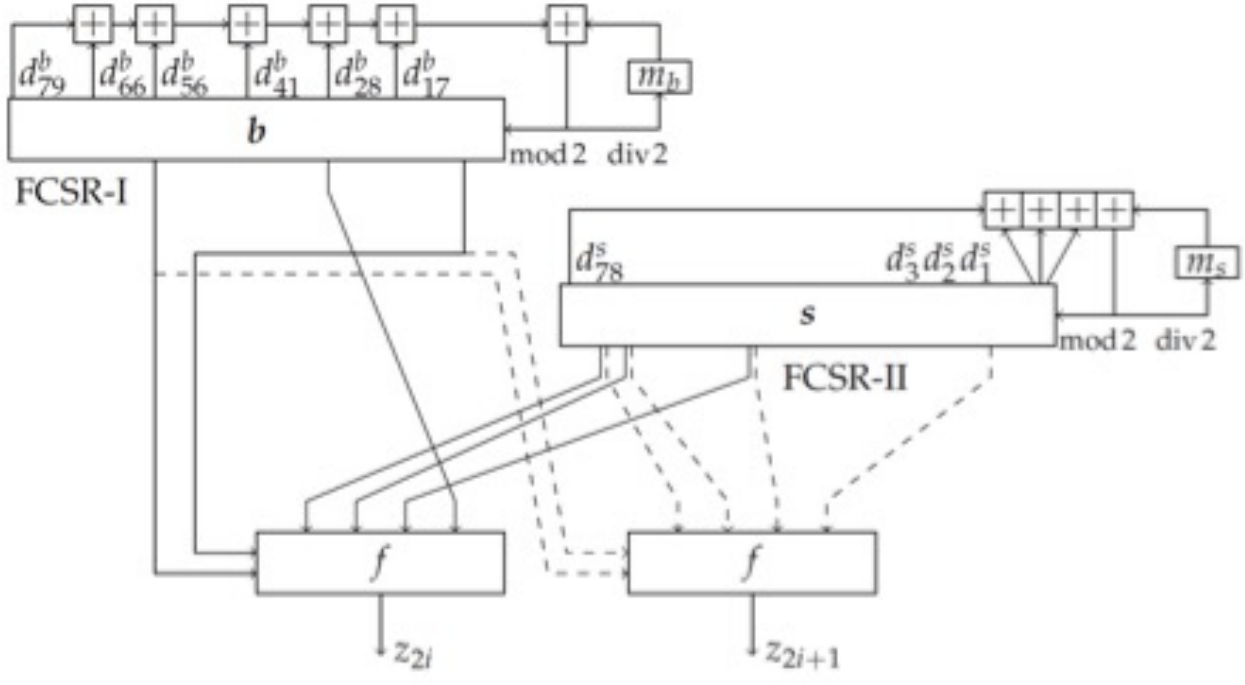
Grain is a bit-oriented synchronous stream cipher, whose design is very easy to implement in hardware and requires only small chip area. It consists of two 80-bit shift registers where one has a linear feedback (LFSR) and the other a nonlinear feedback (NFSR).



This output is used during the initialization phase as additional feedback to LFSR and NFSR (grey lines in the figure indicate this feedback). During normal operation this value is used as key stream output.

::.. Streaming Ciphers (5/6)

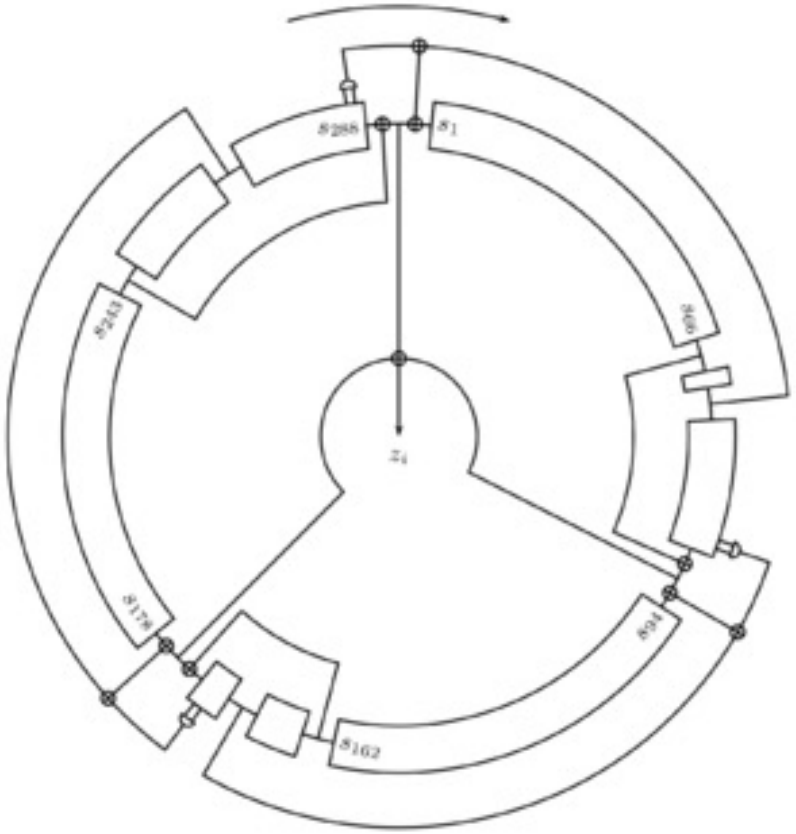
BEAN has some superficial similarities with Grain: the NFSR and the LFSR are replaced by two FCSRs. LFSR in Grain is used to provide large period and to guarantee random-like properties while the NFSR is used to provide nonlinearity. An Fibonacci FCSR combines both these properties, while still being efficient in hardware.



BEAN has two shift register components, both providing non-linearity, large period and random-looking sequences.

::.. Streaming Ciphers (6/6)

Trivium's 288-bit internal state consists of three shift registers of different lengths. At each round, a bit is shifted into each of the three shift registers using a non-linear combination of taps from that and one other register; one bit of output is produced.

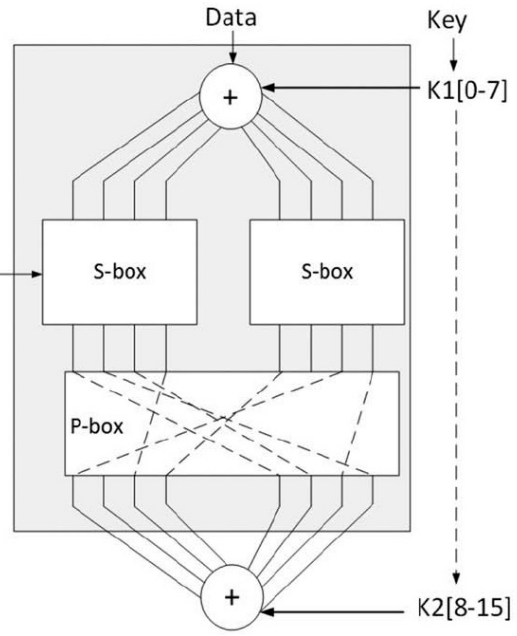


To initialize the cipher, the key and additional input are written into two of the shift registers, with the remaining bits starting in a fixed pattern; the cipher state is then updated 1152 times, so that every bit of the internal state depends on every bit of the key and of the IV in a complex nonlinear way.

::.. Block Ciphers (1/7)

A replacement for AES for lightweight cryptography is PRESENT, which uses smaller block sizes and the potential for smaller keys (such as for an 80-bit key). It users either an 80-bit (10 hex characters) or a 128-bit encryption, and operates on 64-bit blocks and with a **Substitution–Permutation Network** (SPN).

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S[x]$	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

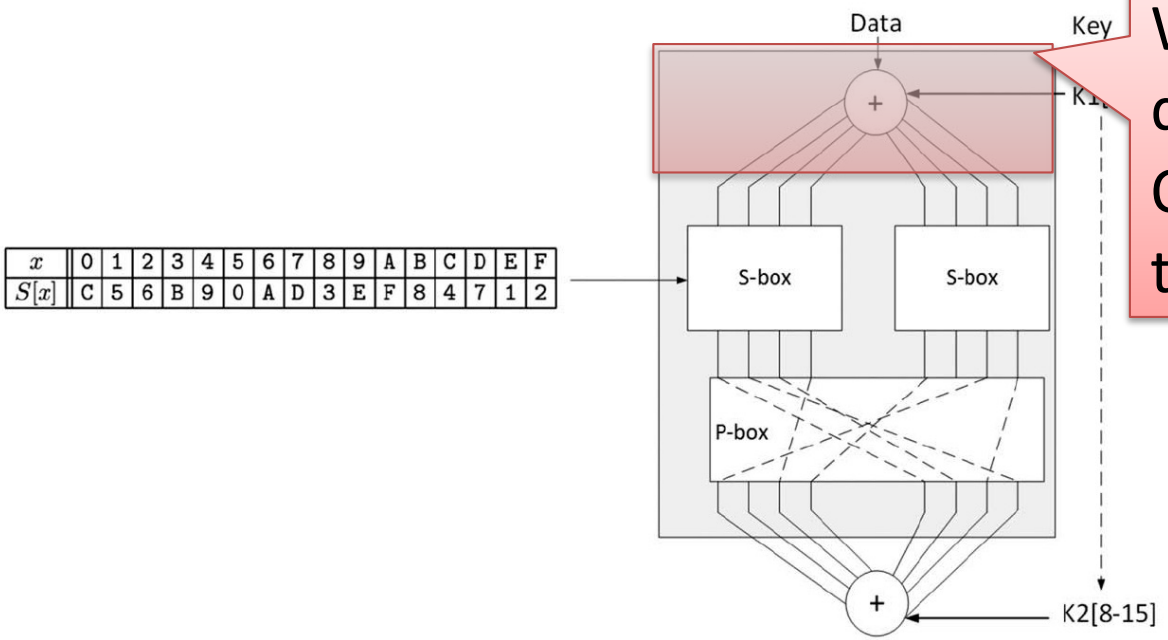


It operates on blocks and apply a key and then use a number of rounds with substitution boxes (S-boxes) and permutation boxes (P-boxes). The decryption process is then the reverse of the encryption rounds.

::.. Block Ciphers (1/7)

A replacement for AES for lightweight cryptography is PRESENT, which uses smaller block sizes and the potential for smaller keys (such as for an 80-bit key). It users either an 80-bit (10 hex characters) or a 128-bit encryption, and operates on 64-bit blocks and with a **Substitution–Permutation Network** (SPN).

It operates on blocks and



x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S[x]$	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

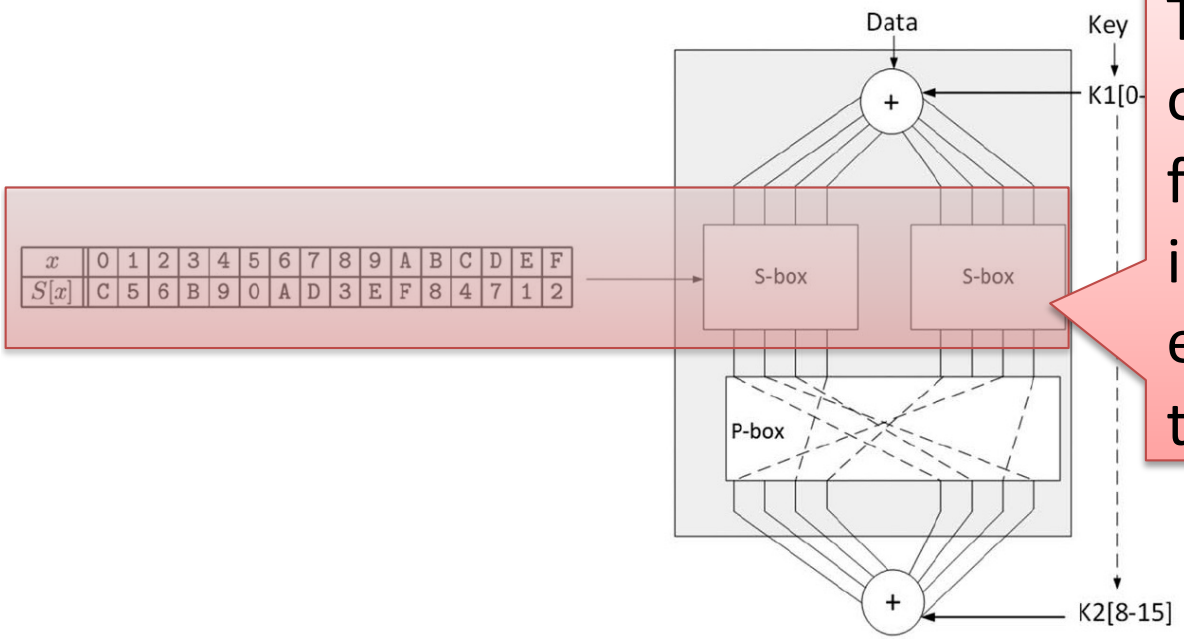
Within a single round, 8-bits of data is entered, and then EX-OR with the first eight bits of the key.

boxes (P-boxes). The decryption process is then the reverse of the encryption rounds.

::.. Block Ciphers (1/7)

A replacement for AES for lightweight cryptography is PRESENT, which uses smaller block sizes and the potential for smaller keys (such as for an 80-bit key). It users either an 80-bit (10 hex characters) or a 128-bit encryption, and operates on 64-bit blocks and with a **Substitution–Permutation Network** (SPN).

It operates on blocks and



The output of the XOR the output from this operation is fed into an S-box, which maps in the inputs to the output (for example, 0 × 0 will be mapped to 0xC).

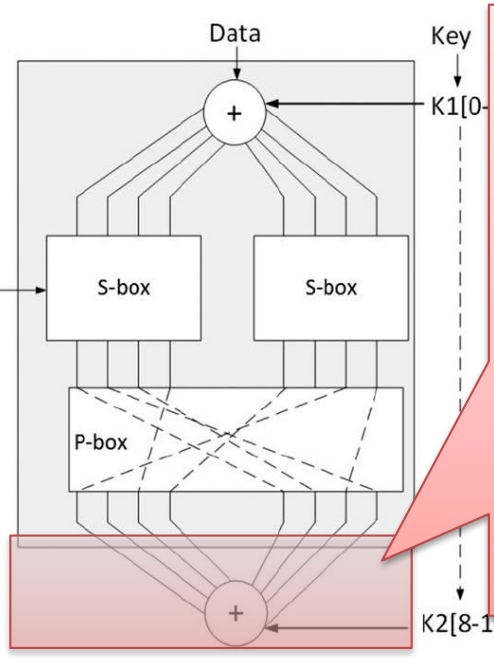
then the reverse of the encryption rounds.

::.. Block Ciphers (1/7)

A replacement for AES for lightweight cryptography is PRESENT, which uses smaller block sizes and the potential for smaller keys (such as for an 80-bit key). It users either an 80-bit (10 hex characters) or a 128-bit encryption, and operates on 64-bit blocks and with a **Substitution–Permutation Network** (SPN).

It operates on blocks and

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S[x]$	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

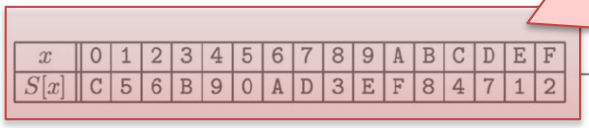


The output of P-box is then fed into the next round, and which will follow the same process, but this time our input is from the previous round and from the next eight bits of the key.

::.. Block Ciphers (1/7)

A replacement for AES for lightweight cryptography is PRESENT, which uses smaller block sizes and the potential for smaller keys (such as for an 80-bit key). It users either an 80-bit (10 hex characters) or a 128-bit encryption, and operates on 64-bit blocks and with a **Substitution–Permutation Network** (SPN).

It operates on blocks and



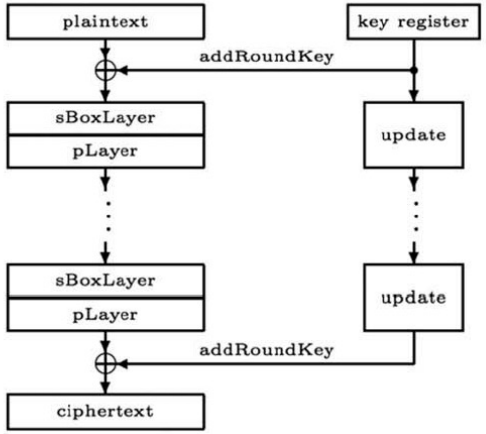
x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S[x]$	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

This substitution should be one-to-one, to ensure invertability (hence decryption). An S-box is usually not simply a permutation of the bits. Rather, a good S-box will have the property that changing one input bit will change about half of the output bits (or an avalanche effect).

encryption rounds.

::.. Block Ciphers (2/7)

```
generateRoundKeys()
for i = 1 to 31 do
  addRoundKey(STATE, K_i)
  sBoxLayer(STATE)
  pLayer(STATE)
end for
addRoundKey(STATE, K_32)
```



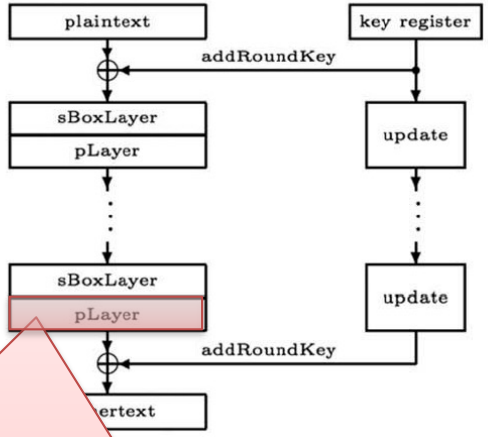
Within PRESENT, the input is a block of 64 bits and an 80-bit or a 128-bit key is applied. Overall, it has 32 rounds.

The key round operation takes part of the key and EXORs it with the data input into the round. It then operates on 4 × 4-bit S-boxes and which considerably cuts down on processing power.

In AES, 16-bit inputs are mapped to 16-bit outputs (0x00 to 0xFF), but for PRESENT, 4-bit values are mapped map onto 16 output values (0x0 to 0xF).

::.. Block Ciphers (2/7)

```
generateRoundKeys()
for i = 1 to 31 do
   addRoundKey(STATE, K_i)
   sBoxLayer(STATE)
   pLayer(STATE)
end for
addRoundKey(STATE, K_32)
```



Within PRESENT, t[illegible] f 64 bits and an 80 bit or a 128-bit key is appli[illegible]

The key round ope[illegible] the data input int[illegible] boxes and which c[illegible]

In AES, 16-bit inp[illegible] 0xFF), but for PRE[illegible] output values (0x0 to 0xF).

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P(i)$	0	16	32	48	1	17	33	49	2	18	34	50	3	19	35	51
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P(i)$	4	20	36	52	5	21	37	53	6	22	38	54	7	23	39	55
i	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$P(i)$	8	24	40	56	9	25	41	57	10	26	42	58	11	27	43	59
i	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$P(i)$	12	28	44	60	13	29	45	61	14	30	46	62	15	31	47	63

::.. Block Ciphers (3/7)

PRESENT is more compact then AES, about 2.5 times smaller.

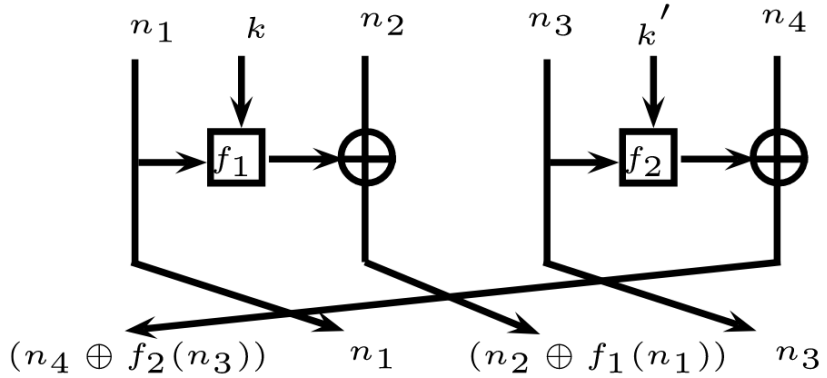
Modules	AES - T	AES - B	Present
Memory [GE]	2040	1678	887
Mix Column [GE]	306	373	0
S-box [GE]	408	233	32
FSM + Rest [GE]	646	317	192
Total Area [GE]	3400	2601	1111
Cycles	1000	226	547
Power [µA]	3.0	3.7	1.34

AES-T
0.35 µm CMOS
Technology of Philips
100kHz @1.5V
Encryption + Decryption

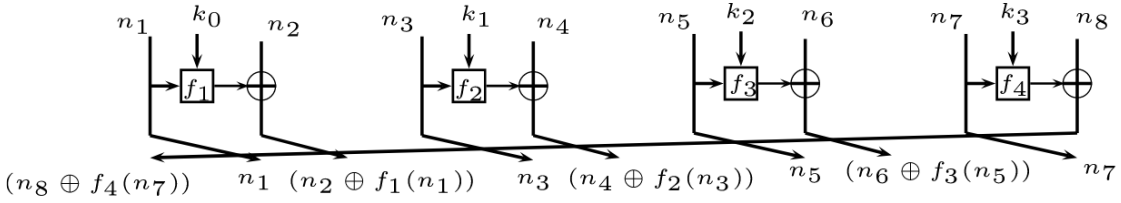
Present and **AES-B**
UMC L180 0.18µm 1P6M
100 kHz @1.8V
Encryption only

::.. Block Ciphers (4/7)

CLEFIA is an encryption algorithm as a safe alternative to AES. The algorithm consists of two component parts: a data processing part and a key scheduling part by using a 4-branch and an 8-branch Type-2 generalized Feistel network.



4-branch type-2 GFN.

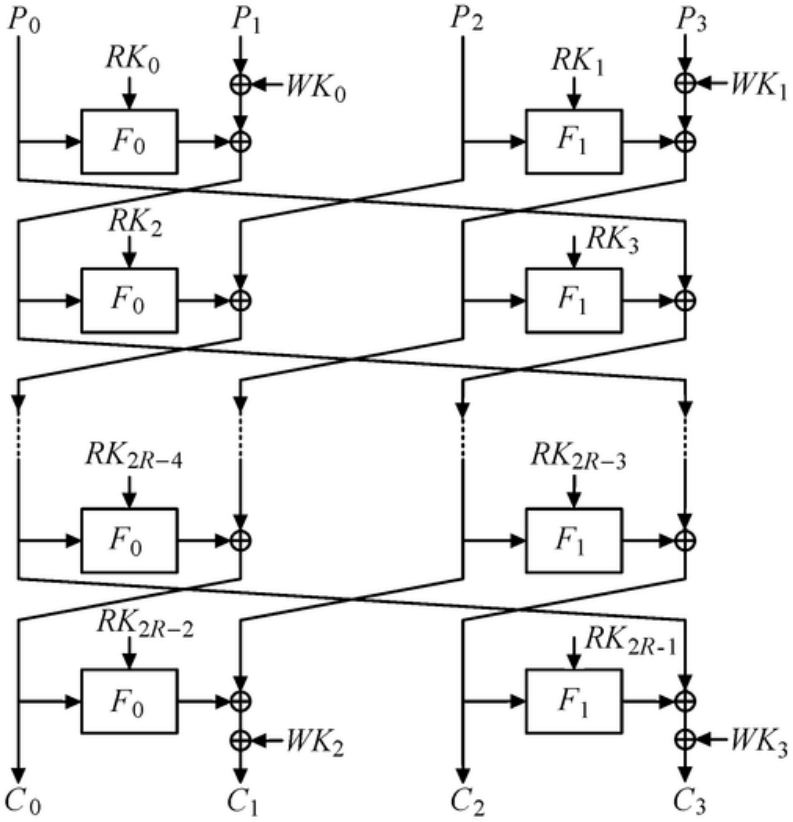


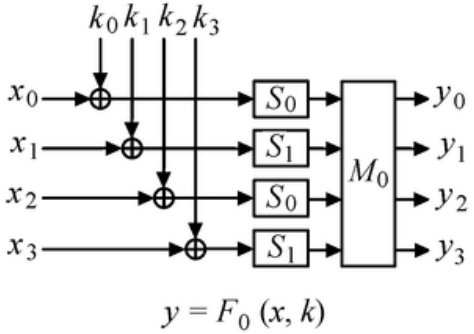
8-branch type-2 GFN.

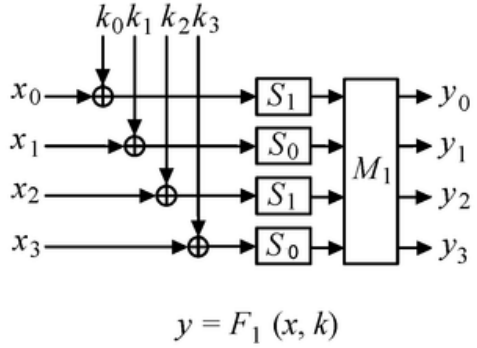
The advantage of this construction is that F-functions are smaller than that in the traditional Feistel structure. Plural F-functions can be processed simultaneously.

::.. Block Ciphers (5/7)

The encryption process uses a 4-branch generalized Feistel structure with two parallel F-functions F0 and F1 per round. Also, there are key whitening parts in the beginning and at the end of the cipher.

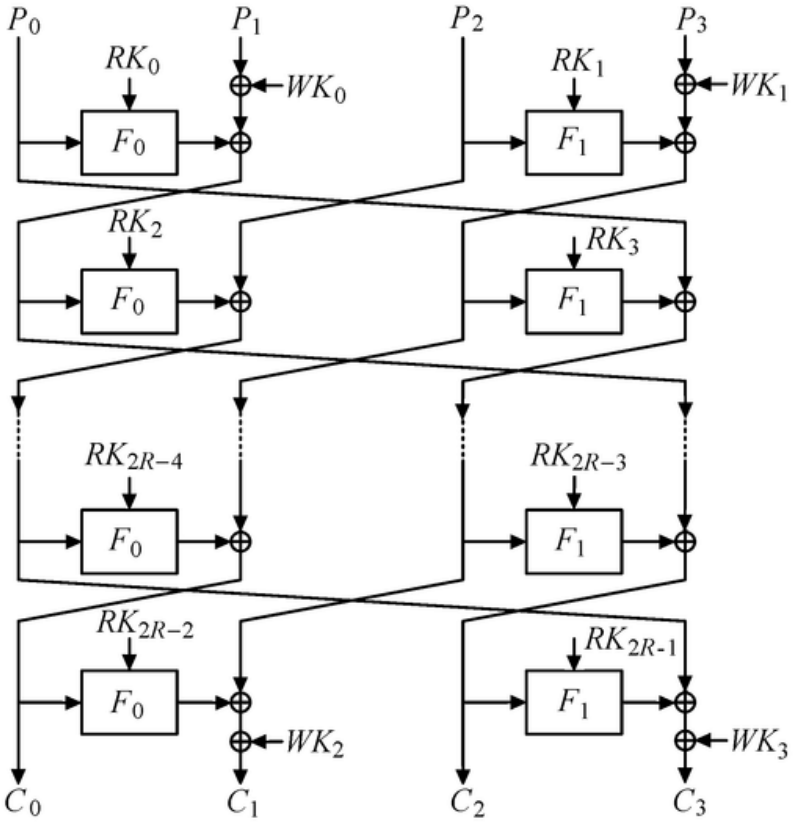


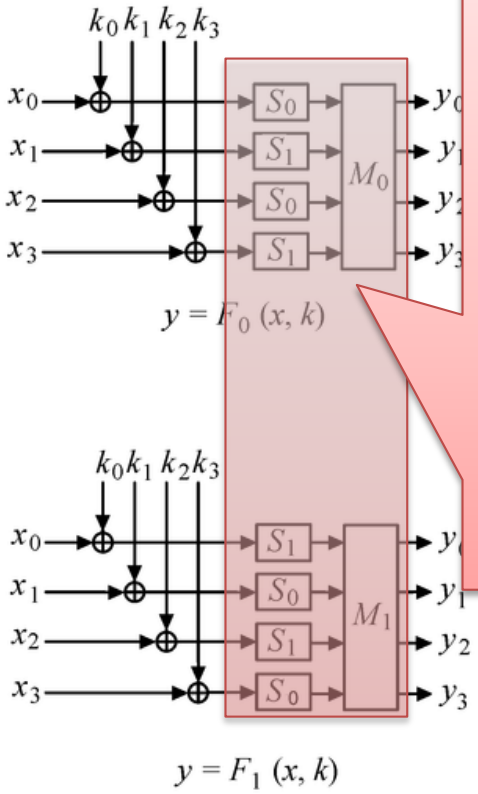




::.. Block Ciphers (5/7)

The encryption process uses a 4-branch generalized Feistel structure with two parallel F-functions F0 and F1 per round. Also, there are key whitening parts in the beginning and at the end of the cipher.



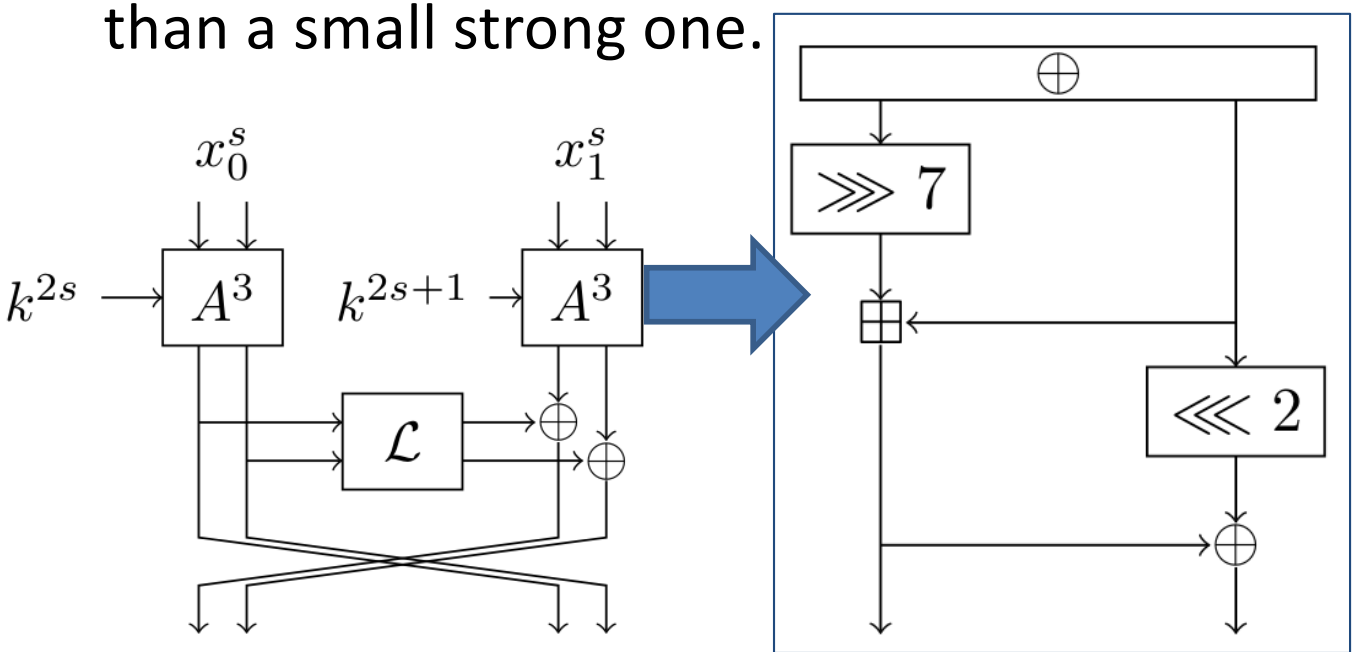


S_0 and S_1 are 8-bit invertible S-boxes, and M_0 and M_1 are two self-inverse 4×4 matrices. The multiplications between these matrices and vectors are performed in GF(2^8)

::.. Block Ciphers (6/7)

SPARX is a family of ARX-based 64- and 128-bit block ciphers. Only addition modulo 2^{16}, 16-bit XOR and 16-bit rotations are needed to implement any version.

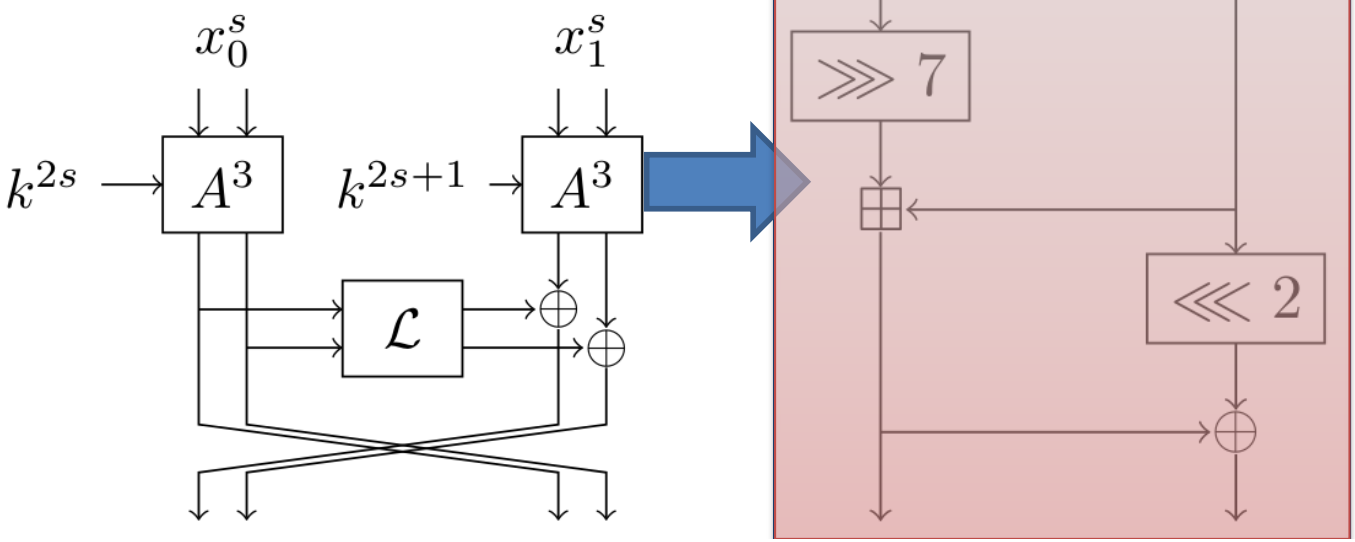
The SPARX ciphers have been designed according to the Long Trail Strategy, which is a counterpart of the Wide-Trail Strategy, suitable for algorithms built using a large and weak S-Box rather than a small strong one.



::.. Block Ciphers (6/7)

SPARX is a family of ARX-based 64- and 128-bit block ciphers. Only addition modulo 2^{16}, 16-bit XOR and 16-bit rotations are needed to implement any version.

The SPARX ciphers have been designed acco
Trail Strategy, which is a counterpart of the
suitable for algorithms built using a large an
than a small strong one.

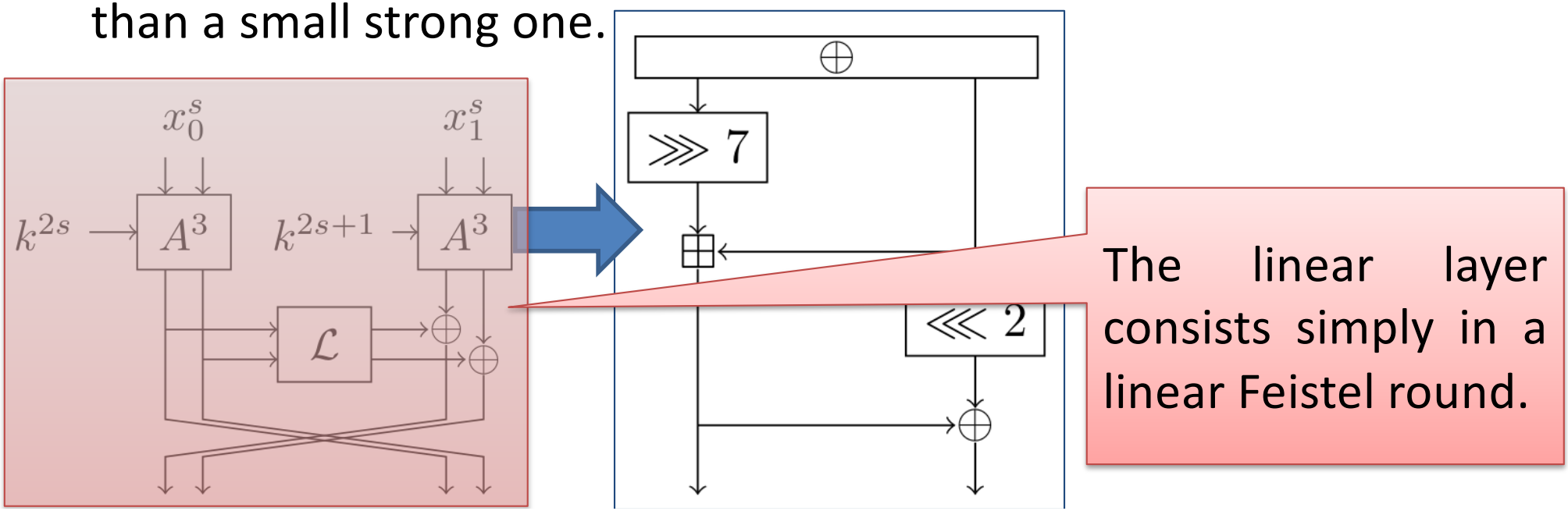


The non-linearity is provided by SPECK-KEY xoring plaintext and the key and rotating 7 bits right of the first part, added to the second, and rotates 2 bits left the second part xored to the resut of the addition.

::.. Block Ciphers (6/7)

SPARX is a family of ARX-based 64- and 128-bit block ciphers. Only addition modulo 2^{16}, 16-bit XOR and 16-bit rotations are needed to implement any version.

The SPARX ciphers have been designed according to the Long Trail Strategy, which is a counterpart of the Wide-Trail Strategy, suitable for algorithms built using a large and weak S-Box rather than a small strong one.

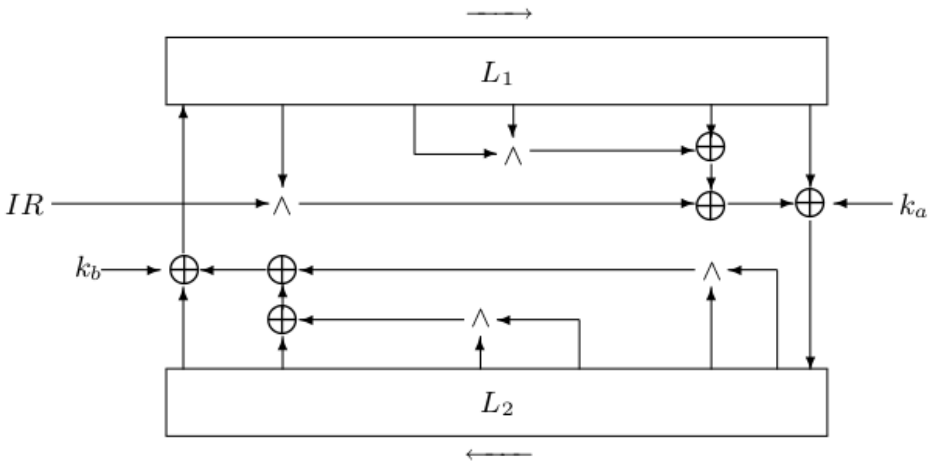


The linear layer consists simply in a linear Feistel round.

::.. Block Ciphers (7/7)

KATAN has been designed to optimize the physical footprint:

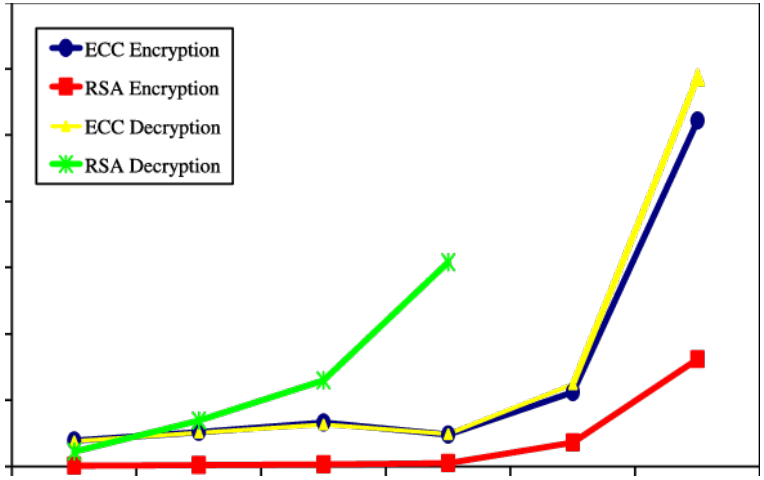
- The plaintext is loaded into two registers (whose lengths depend on the block size).
- Each round, several bits are taken from the registers and enter two nonlinear Boolean functions and proper xor with the key.
- The output of the Boolean functions is loaded to the least significant bits of the registers (after they were shifted).



- To ensure sufficient mixing, 254 rounds of the cipher are executed.

::.. Elliptic Curve Cryptosystems (1/7)

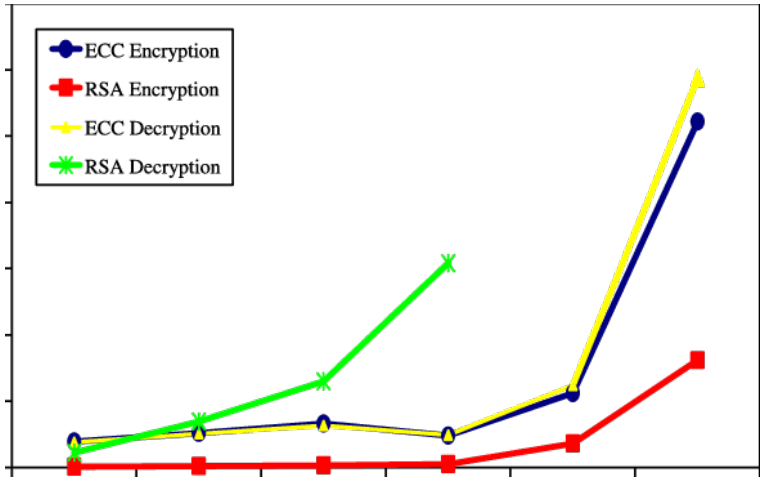
Security Level	RSA Key Size	ECDSA Key Size
80	1024 bits	160–223 bits
112	2048 bits	224–255 bits
128	3072 bits	256–383 bits
192	7680 bits	384–511 bits
256	15360 bits	512+ bits



Elliptic Curve Cryptosystems (ECC) provides similar level of secrecy such as cryptosystems based on Discrete Logarithm Problem (DLP) or an Integer Factorization Problem (IFP) but with shorter key length.

::.. Elliptic Curve Cryptosystems (1/7)

Security Level	RSA Key Size	ECDSA Key Size
80	1024 bits	160–223 bits
112	2048 bits	224–255 bits
128	3072 bits	256–383 bits
192	7680 bits	384–511 bits
256	15360 bits	512+ bits

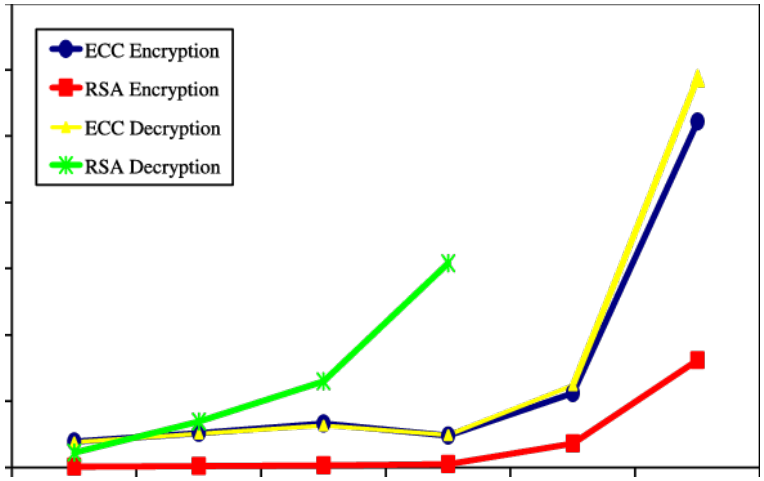


Elliptic Curve Cryptosystems (ECC) provides similar level of secrecy such as cryptosystems based on Discrete Logarithm Problem (DLP) or an Intege Factorization Problem (IFP) but with rter key length.

Given a group G, a generator g of the group and an element h of G, to find the discrete logarithm to the base g of h in the group G.

::.. Elliptic Curve Cryptosystems (1/7)

Security Level	RSA Key Size	ECDSA Key Size
80	1024 bits	160–223 bits
112	2048 bits	224–255 bits
128	3072 bits	256–383 bits
192	7680 bits	384–511 bits
256	15360 bits	512+ bits

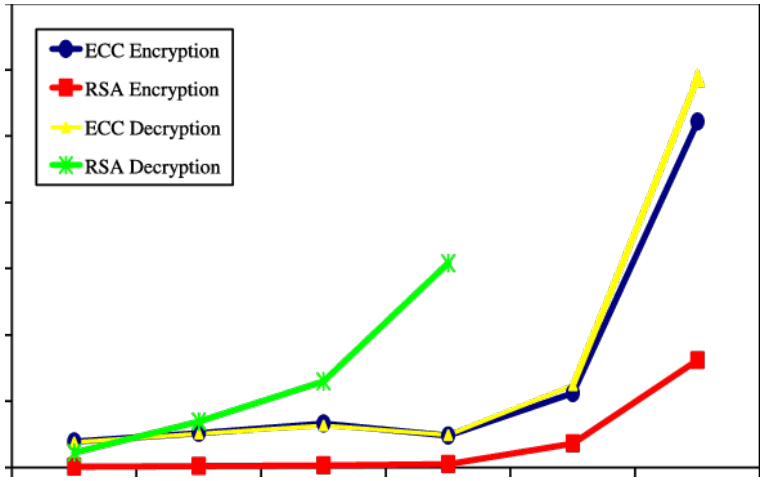


Elliptic Curve Cryptosystems (ECC) provides similar level of secrecy such as cryptosystems based on Discrete Logarithm Problem (DLP) or an Integer Factorization Problem (IFP) but with sh rter key length.

It consists into looking for the decomposition of a composite number into a product of smaller integers. The hardest instance is considering semiprimes, the product of two prime numbers, and solving its factorization.

::.. Elliptic Curve Cryptosystems (1/7)

Security Level	RSA Key Size	ECDSA Key Size
80	1024 bits	160–223 bits
112	2048 bits	224–255 bits
128	3072 bits	256–383 bits
192	7680 bits	384–511 bits
256	15360 bits	512+ bits



Elliptic Curve Cryptosystems (ECC) provides similar level of secrecy such as cryptosystems based on Discrete Logarithm Problem (DLP) or an Integer Factorization Problem (IFP) but with shorter key length.

An elliptic-curve is a group (E, $\oplus$) defined over a field (F, +, x). Since E is a group, the operations "$\oplus$" and "$\ominus$" over E can be done by using operations in the underlying field F.

An elliptic-curve E defined over a prime field F_{11} as follows:

$$E/F_{11}: y^2 = x^3 + 4x + 3,$$

where $F_{11} = \{0, 1, \ldots, 10\}$ is a prime field with 11 elements.

::.. Elliptic Curve Cryptosystems (2/7)

Let us consider an addition and multiplication tables over F_{11}:

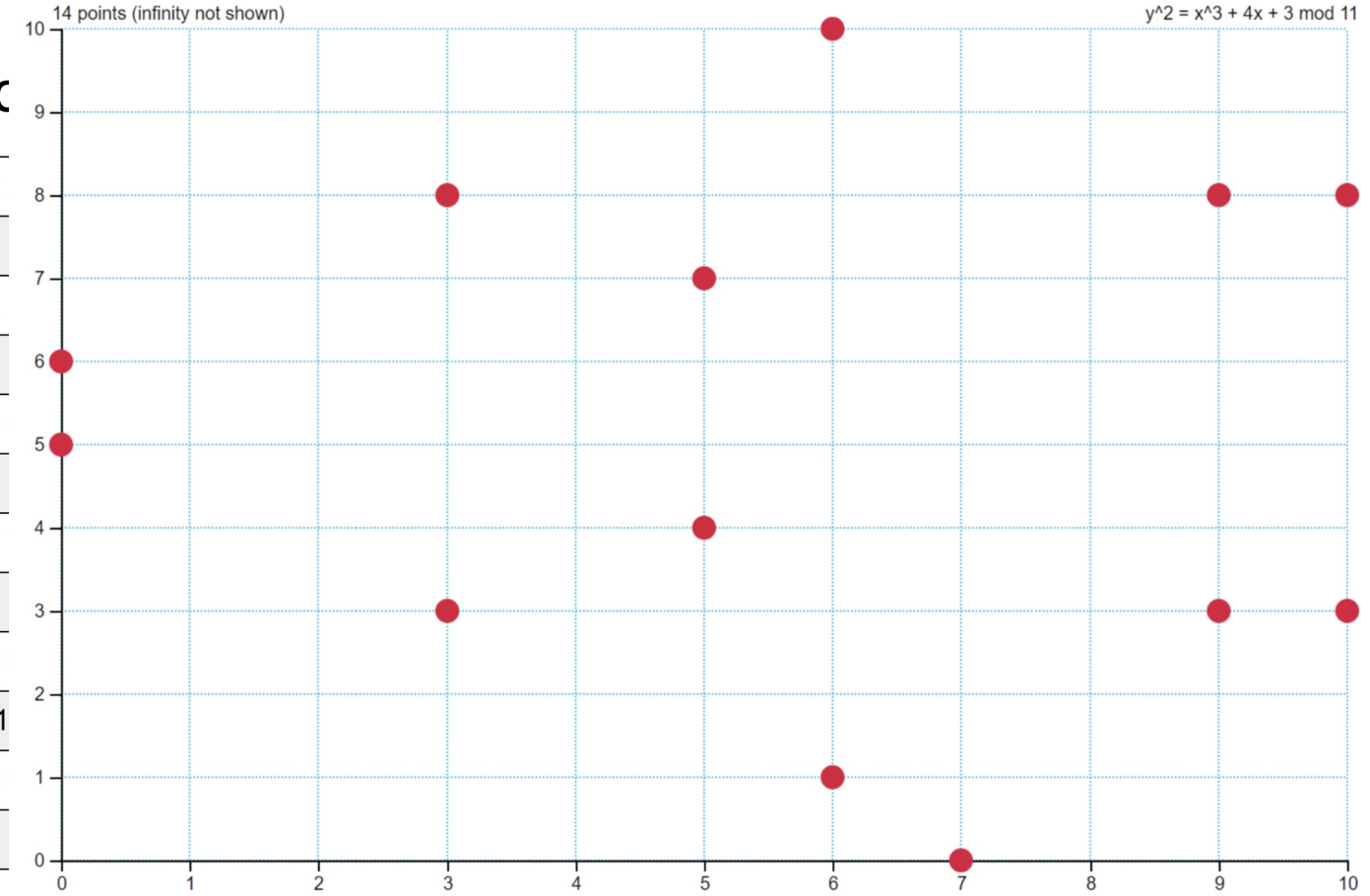
+	0	1	2	3	4	5	6	7	8	9	10
0	0	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10	0
2	2	3	4	5	6	7	8	9	10	0	1
3	3	4	5	6	7	8	9	10	0	1	2
4	4	5	6	7	8	9	10	0	1	2	3
5	5	6	7	8	9	10	0	1	2	3	4
6	6	7	8	9	10	0	1	2	3	4	5
7	7	8	9	10	0	1	2	3	4	5	6
8	8	9	10	0	1	2	3	4	5	6	7
9	9	10	0	1	2	3	4	5	6	7	8
10	10	0	1	2	3	4	5	6	7	8	9

×	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	10
2	0	2	4	6	8	10	1	3	5	7	9
3	0	3	6	9	1	4	7	10	2	5	8
4	0	4	8	1	5	9	2	6	10	3	7
5	0	5	10	4	9	3	8	2	7	1	6
6	0	6	1	7	2	8	3	9	4	10	5
7	0	7	3	10	6	2	9	5	1	8	4
8	0	8	5	2	10	7	4	1	9	6	3
9	0	9	7	5	3	1	10	8	6	4	2
10	0	10	9	8	7	6	5	4	3	2	1

By conducting an exhaustive search, 13 solutions over F_{11} can be found for the equation of the elliptic-curve E, namely (0, 5), (0, 6), (3, 3), (3, 8), (5, 4), (5, 7), (6, 1), (6, 10), (7, 0), (9, 3), (9, 8), (10, 3) and (10, 8).

::.. Elliptic Curve Cryptosystems (2/7)

Let us c



F_{11}:

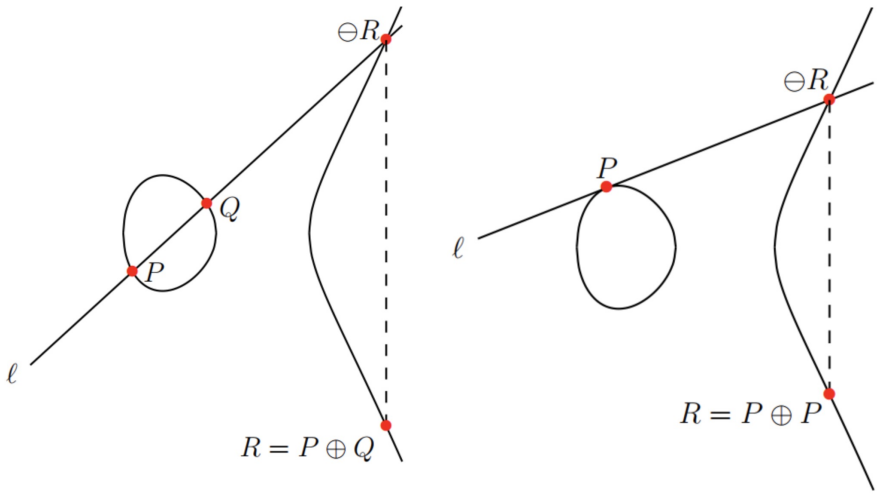
+	0	1	
0	0	1	
1	1	2	
2	2	3	
3	3	4	
4	4	5	
5	5	6	
6	6	7	
7	7	8	
8	8	9	1
9	9	10	
10	10	0	

7	**8**	**9**	**10**
0	0	0	0
7	8	9	10
3	5	7	9
10	2	5	8
6	10	3	7
2	7	1	6
9	4	10	5
5	1	8	4
1	9	6	3
8	6	4	2
4	3	2	1

By conducting an exhaustive search, 13 solutions over F_{11} can be found for the equation of the elliptic-curve E, namely (0, 5), (0, 6), (3, 3), (3, 8), (5, 4), (5, 7), (6, 1), (6, 10), (7, 0), (9, 3), (9, 8), (10, 3) and (10, 8).

::.. Elliptic Curve Cryptosystems (3/7)

All of these 13 solutions and a special point O (so-called point at infinity) form a group of 14 elements.



Given two points P and Q in $E(F_{11})$, $P \oplus Q = R$ can be computed following the so-called "chord-and-tangent" rule.

$P \oplus Q$ (resp. $P \oplus P$) is computed by drawing a chord (resp. tangent) line through those two (resp. one) points, which intersects with the elliptic-curve at the point $\ominus R$. Flipping this point over x-axis gets the point $R = P \oplus Q$ (resp. $R = P \oplus P$).

::.. Elliptic Curve Cryptosystems (3/7)

Mathematically speaking, the rules for adding two points P and Q over an elliptic-curve $E(F_{11})$ are defined as follows:

- $P \oplus O = P$, where O is the point-at-infinity.
- If P = (Xp, Yp), then $P \oplus (Xp, -Yp) = O$. The point (Xp, -Yp) is the negative of P, $\ominus P$. For example, in $E(F_{11})$ if P = (5, 4), then $\ominus P$ = (5, -4) = (5, 7) since $-4 \equiv 7 \bmod 11$, which is also in $E(F_{11})$.
- If P = (Xp, Yp) and Q = (Xq, Yq) with $P \neq \ominus Q$, then $R = P \oplus Q$ = (Xr, Yr) is in $E(F_{11})$ and determined by the following formulae: $Xr = (\lambda^2 - Xp - Xq) \bmod 11$ and $Yr = [\lambda(Xp - Xr) - Yp] \bmod 11$, where $\lambda = (Yp - Yq) / (Xp - Xq) \bmod 11$, if $P \neq Q$. In the case of P = Q, $\lambda = (3Xp^2 + 4) / 2Yp \bmod 11$.
- Multiplication by an integer k is defined by repeated addition, which is usually called a scalar multiplication. For example, $5P = P \oplus P \oplus P \oplus P \oplus P$.

::.. Elliptic Curve Cryptosystems (4/7)

Any public key cryptosystem is based on difficult computational problems. Elliptic-curve cryptosystem is no exception.

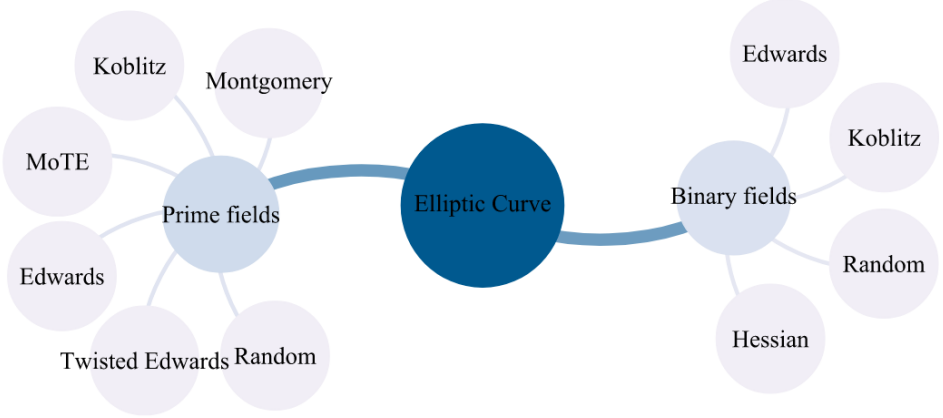
Consider the equation Q = kP, where P and Q are two points over an elliptic-curve $E(F_p)$ with p being a sufficiently large prime (e.g., ~2^160). While it is relatively easy to calculate Q given k and P, it is extremely difficult to determine k given Q and P. This is called **elliptic-curve discrete logarithm problem** (ECDLP).

From $E(F_{11})$, the discrete logarithm problem can be easily solved via a table-lookup. Given that P = (3, 3), we can compute kP, k = 0, …, 14, using the point addition formulae and put the results in a table as follows:

0P = *O*	1P = (3, 3)	2P = (10, 3)	3P = (9, 8)	4P = (0, 5)
5P = (6, 10)	6P = (5, 7)	7P = (7, 0)	8P = (5, 4)	9P = (6, 1)
10P = (0, 6)	11P = (9, 3)	12P = (10, 8)	13P = (3, 8)	14P = *O*

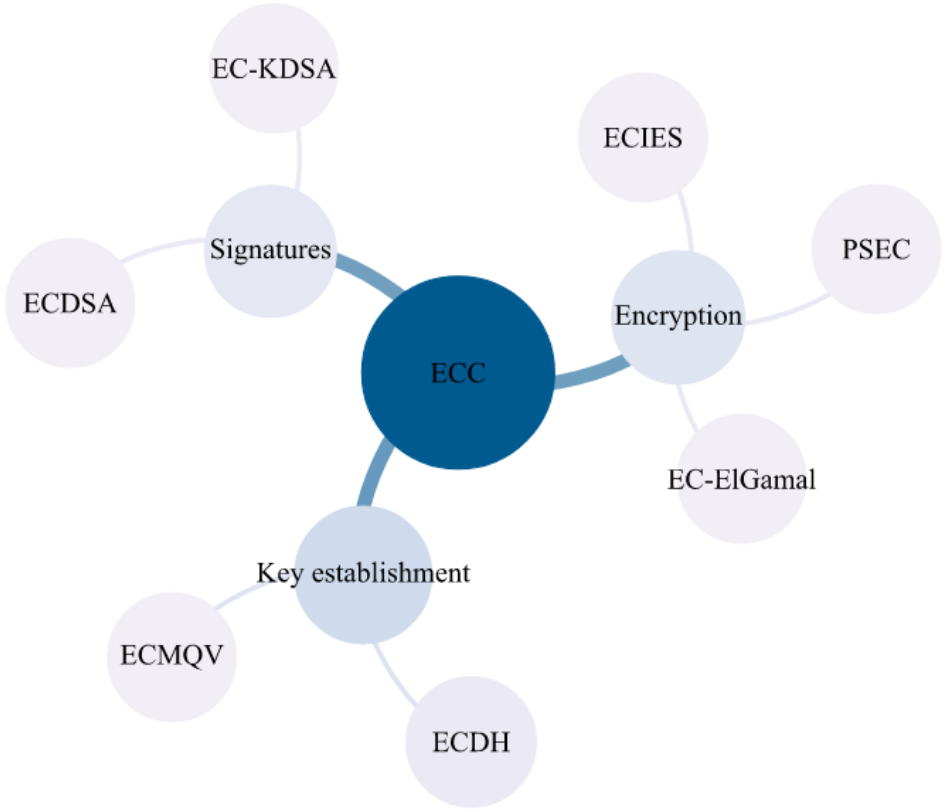
::.. Elliptic Curve Cryptosystems (5/7)

Given any Q in $E(F_{11})$, we can find the corresponding k such that Q = kP. For example, for Q = (5, 4), we know that k = 8 by checking the previous table. This method is not applicable when p is a large prime. Therefore, the ECDLP ensures the security of elliptic-curve cryptosystems in practice.



The selection of the curve equation is extremely important in terms of latency and provided security level. Families of elliptic curves defined over prime and binary fields can be used, where each family has a corresponding curve model which represents its curves.

::.. Elliptic Curve Cryptosystems (6/7)



The most important security services provided by ECC include key establishment, confidentiality, integrity, and authentication.

The encryption by means of ECC is based on the solution of the Elliptic Curve Discrete Logarithm Problem (ECDLP).

Given an elliptic curve E defined over a finite field F_q, a point $P \in E(F_q)$ of order n, and a point $Q \in \langle P \rangle$, find the integer $l \in [0,n-1]$ such that $Q = l \cdot P$. The integer l is called the discrete logarithm of Q to the base P, denoted $l = \log_P Q$.

::.. Elliptic Curve Cryptosystems (6/7)

An Encryption Scheme based on elliptic curves is a variant of the ElGamal public-key encryption scheme, standardized in ANSI X9.63 and ISO/IEC 15946-3, and is in the IEEE P1363a draft standard.

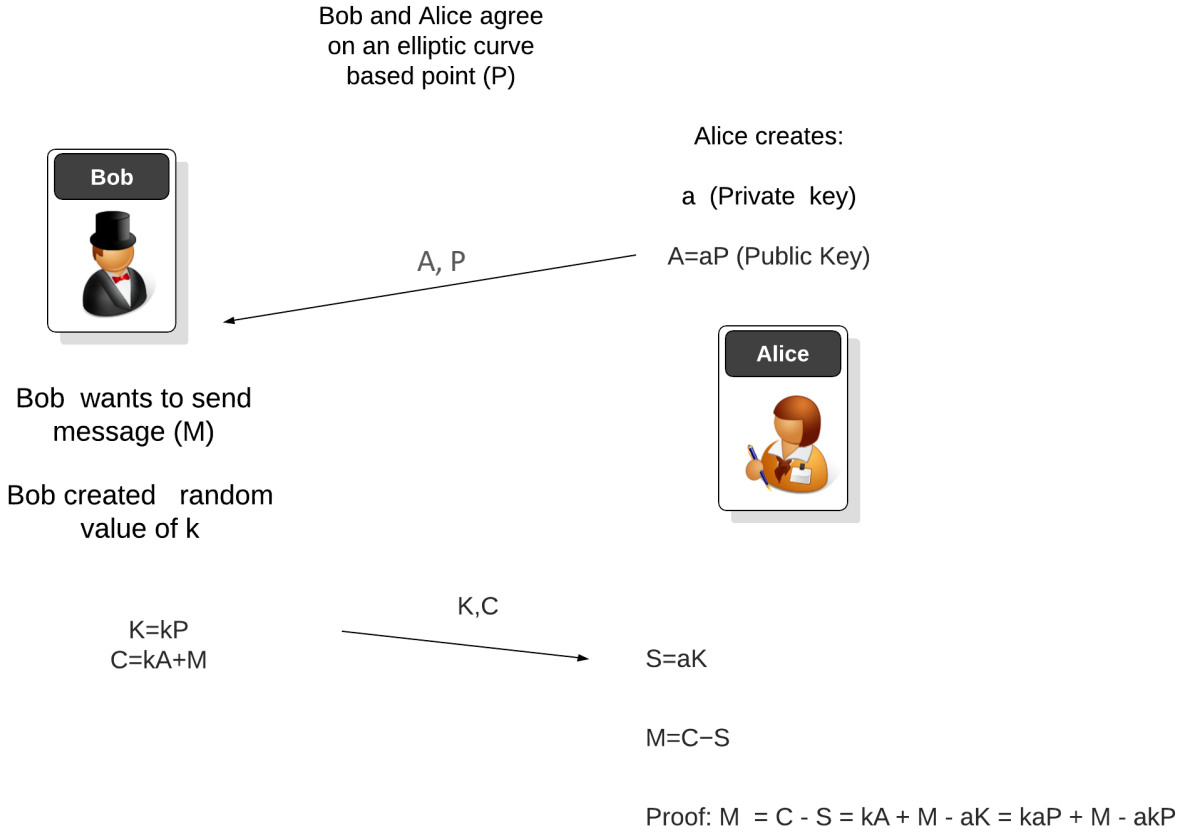
::.. Elliptic Curve Cryptosystems (6/7)

An Encryption Scheme based on elliptic curves is a variant of the ElGamal public-key encryption scheme, standardized in ANSI

1. Bob generates public and private key :
   - Bob chooses a very large number **q** and a cyclic group $\mathbf{F}_q$.
   - From the cyclic group $\mathbf{F}_q$, he choose any element **g** and an element **a** such that gcd(a, q) = 1.
   - Then he computes $h = g^a$.
   - Bob publishes **F**, $\mathbf{h = g^a}$, **q** and **g** as his public key and retains **a** as private key.
2. Alice encrypts data using Bob's public key :
   - Alice selects an element **k** from cyclic group **F** such that gcd(k, q) = 1.
   - Then she computes $p = g^k$ and $s = h^k = g^{ak}$.
   - She multiples s with M.
   - Then she sends $(p, M*s) = (g^k, M*s)$.
3. Bob decrypts the message :
   - Bob calculates $s' = p^a = g^{ak}$.
   - He divides M*s by s' to obtain M as $s = s'$.

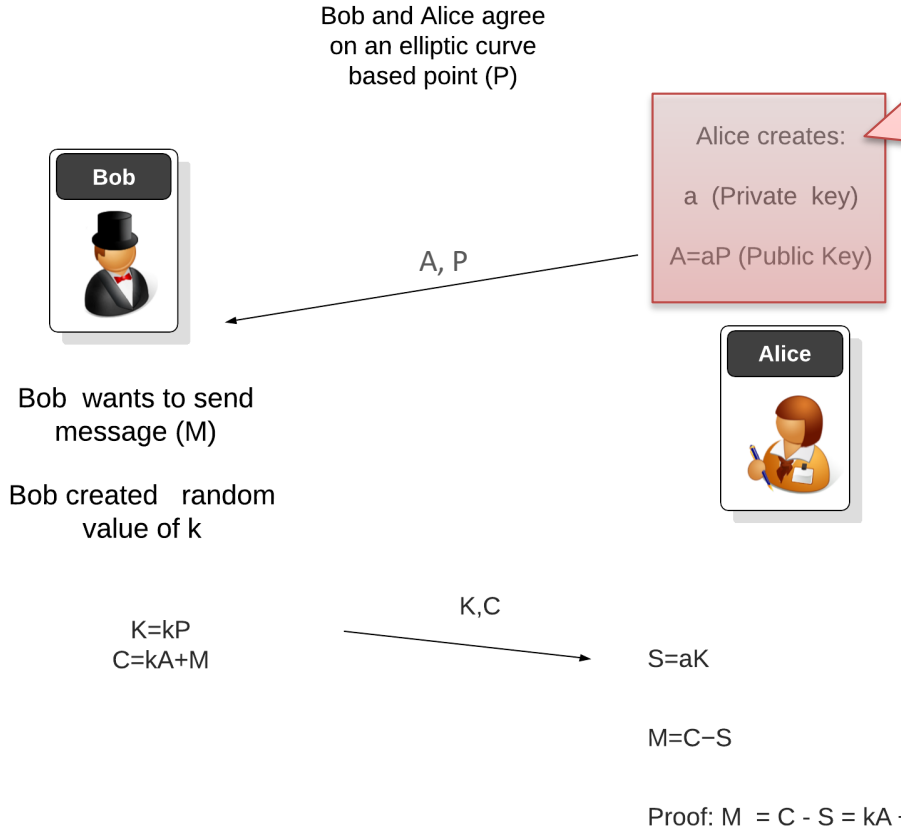
::.. Elliptic Curve Cryptosystems (6/7)

An Encryption Scheme based on elliptic curves is a variant of the ElGamal public-key encryption scheme, standardized in ANSI X9.63 and ISO/IEC 15946-3, and is in the IEEE P1363a draft standard.



::.. Elliptic Curve Cryptosystems (6/7)

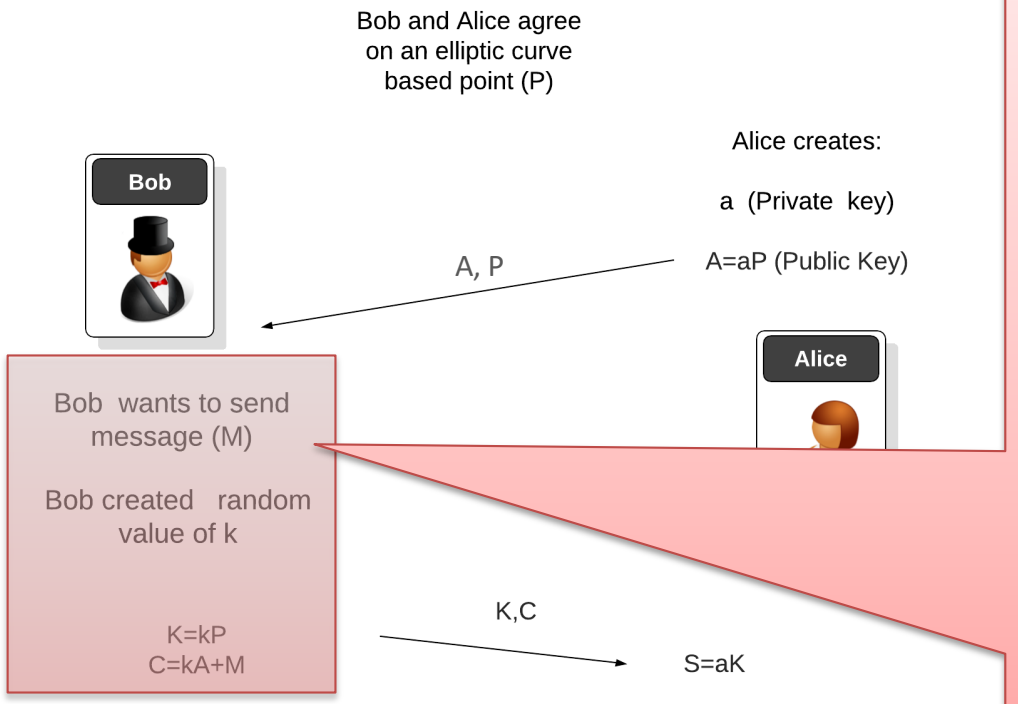
An Encryption Scheme based on elliptic curves is a variant of the ElGamal public-key encryption scheme, standardized in ANSI X9.63 and ISO/IEC 15946-3, and is in the IEEE P1363a draft standard.



Alice first creates a private key (a), which is a random scalar value, and a public key (aP), which is a point on the elliptic curve.

::.. Elliptic Curve Cryptosystems (6/7)

An Encryption Scheme based on elliptic curves is a variant of the ElGamal public-key encryption scheme, standardized in ANSI X9.63 and ISO/IEC 15946-3, and is in the IEEE P1363a draft standard.



If Bob wants to send Alice an encrypted message (M), he creates a random value (k) and uses her public key (A) to produce a cipher message of:

$K=kP$

and the another one with:

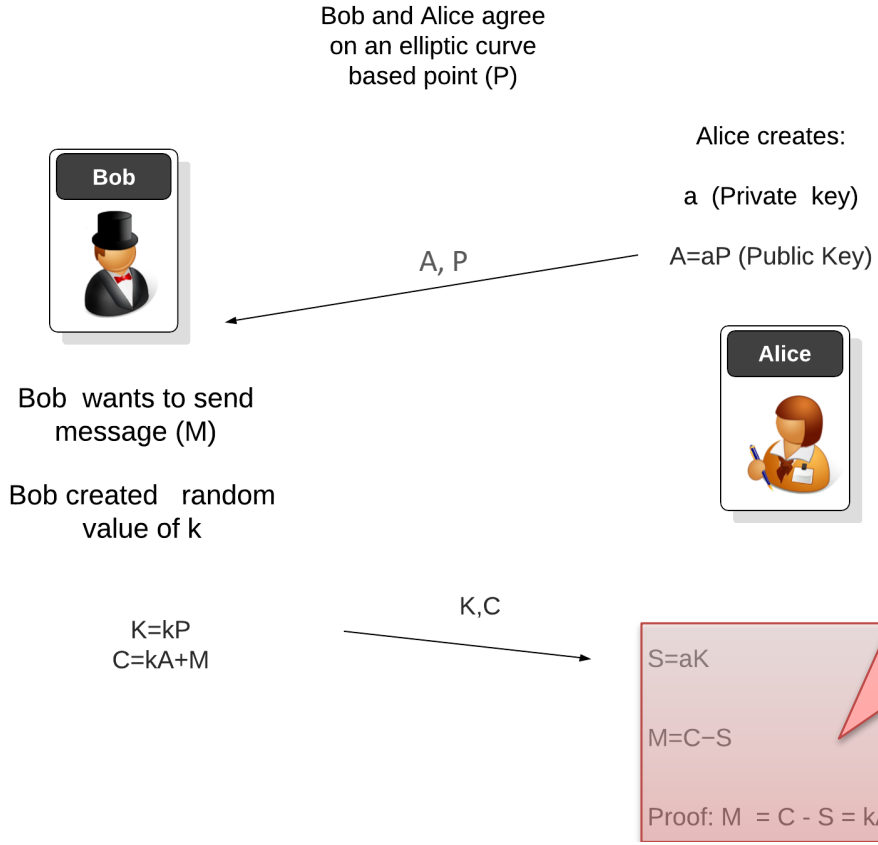
$C=kA+M$

where M is matched to a point on the elliptic curve.

Proof: $M = C - S = kA + M - aK = kaP + M - akP$

::.. Elliptic Curve Cryptosystems (6/7)

An Encryption Scheme based on elliptic curves is a variant of the ElGamal public-key encryption scheme, standardized in ANSI X9.63 and ISO/IEC 15946-3, and is in the IEEE P1363a draft standard.



Now Alice receives (K) and (C), and computes:

$S=aK$

and then computes the message with:

$M=C-S$

As C and S will be points on the elliptic curve, this will be done with a point subtraction operation.

::.. Elliptic Curve Cryptosystems (7/7)

The ECC based cryptographic primitives usually require three kinds of time-consuming operations:

- Fixed point multiplication P ($k \cdot P$),
- Random point multiplication Q ($k \cdot Q$),
- double base scalar multiplication $k \cdot P + l \cdot Q$.

Since such operations are iteratively evaluated via a sequence of point doublings and point additions, it is natural to accelerate these computation by reducing the number of point doublings or point additions as minimal as possible.