M2 Cybersecurity (Univ. Grenoble Alpes/Grenoble INP)

Cryptographic Mechanisms (Part 1)

v1.00, September 2016

Philippe Elbaz-Vincent

Website of the course

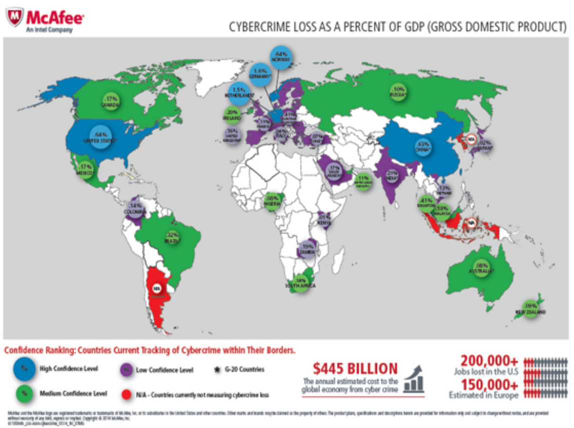
http ://www-fourier.ujf-grenoble.fr/∼pev/cybersecurity

login : cysec2016

passwd : Rijndael_is_cool !

Then go in the folder **Crypto**.

Motivations ?=? $$$



Fact (McAfee/CSIS report 2014) : *"Economic Impact of Cybercrime Estimated at $445 billion Worldwide, and between 15 percent and 20 percent of the value created by the Internet".*

Short syllabus for Crypto lectures

To give a rough idea of the roadmap...

- Background concepts, RSA and AES as a fundamental example of asymmetric/symmetric ciphers (from textbook to real-life)
- DLP over finite fields [short lecture]
- Elliptic Curve Cryptography (ECC) [extended lectures and training]
- PostQuantum Cryptography [intro lectures]
- Random Number Generators (RNGs) and generation of parameters.
- Whitebox cryptography (overview)

Grading : 1 classroom assessment (as MCQ) + 1 practical work + 1 exam

Exam will have mainly a MCQ (basic skill and high level skill).

Some references

- Nigel SMART ; Cryptography Made Simple, Springer Verlag 2015.
- Wenbo MAO ; Modern Cryptography : Theory and Practice, Prentice Hall 2003.
- Jonathan KATZ, Yehuda LINDEL ; Introduction to Modern Cryptography, Chapman and Hall/CRC 2014.
- Colin BOYD, Anish MATHURIA ; Protocols for Authentication and Key Establishment, Springer Verlag 2010.
- Niels Ferguson, Bruce Schneier, Tadayoshi Kohno ; Cryptography Engineering : Design Principles and Practical Applications, 2010.

The Main Crypto-standards (cf. on the web site)

- The RGS 2.0 from ANSSI (2014),
- Standards from the NIST (signatures, hash function, ciphers, key establishment protocols, RNG,...) and the FIPS certifications,
- Standards from ANSI and ISO/IEC,
- The family IEEE 1363-2000 (and their updates),
- The family PKCS (Public Key Cryptography Standard), initiated by RSA, and continuated by some consortium (e.g., OASIS for PKCS #11, one of the main standard for crypto hardware),
- EMVCo (bank and payments, see PKI lectures).

Important remark !

☞ For all those lectures, we will speak of *cryptography* based on **computational assumptions**.

Those concern the most frequent cryptosystems in the real life. But there are other types of cryptosystems, such as *quantum cryptography* (which is a reality !), which are based on **hypothesis from physics** (usually quantum mechanic or general relativity).

As a consequence you need to understand complexity and computational capabilities...

Orders of magnitude (in 2^n scale)

n	2^n	**Order of magnitude**
32	2^{32}	Number of humans on Earth
46	2^{46}	Distance Earth-Sun in *millimeters*
46	2^{46}	Number of op. done by a mono-core CPU at 1 Ghz
55	2^{55}	Number of op. done in one year by a mono-core CPU at 1 Ghz
63	2^{63}	Number of op. done in one year by a server with quadri-CPUs 22 cores at 2.2 Ghz
82	2^{82}	Mass of the Earth in *kilograms*
89	2^{89}	Number of op. done in 13,8 Billions years (estimated age of the Universe) assuming 1 billion of op. per seconde
155	2^{155}	Number of water molecules on Earth
256	2^{256}	Number of electrons in the Universe

The four essential keywords of cryptography

In the "real life", a cryptosystem should allow us to perform the following abilities :

- Confidentiality : it should be very difficult to break a message.
- Authentication : it should be possible for the receiver of a message to ensure the origin of the message. A third party should not be able to fake an identity.
- Integrity : the receiver should be able to ensure that the content has not been modified during the transmission. A third party should not be able to substitute part of a genuine message without being detected.
- Non repudiation : the sender (or the receiver) should not be able to deny the fact that he/she has sent (or received) a message.

Remark : As we can see, the three above constraints are of main importance from a law point of view.

Algorithms and keys

An algorithm of cryptography (or cryptographic method) denotes the set of mathematical functions used for the ciphering and the deciphering. Such functions are usually viewed as a single function (or algorithm).

While in the prehistory of the cryptography, the security of a cryptosystem was solely based on the fact that the cryptographic method was secret. *It is only toward the end of the 19th century which was pointing out the fact that the security must not depend on the cryptographic method*

The articles of Auguste Kerckhoffs (« La cryptographie militaire », Journal des sciences militaires, vol. IX, pp. 5–38, Janvier 1883, pp. 161–191, Février 1883) set the background for modern cryptography

Auguste Kerckhoffs 1835–1903

Kerkhoffs' Laws

Here are the (translation of the) six principles of practical cipher design of Kerckhoffs (originally for military cryptosystem) :

1. The system should be, if not theoretically unbreakable, unbreakable in practice ;
2. **The design of a system should not require secrecy and compromise of the system should not inconvenience the correspondents** ;
3. The key should be memorable without notes and should be easily changeable ;
4. The cryptograms should be transmittable by telegraph ;
5. The apparatus or documents should be portable and operable by a single person ;
6. The system should be easy, neither requiring knowledge of a long list of rules nor involving mental strain.

The two family of cryptosystems

- Symmetric cipher
- Asymmetric cipher

Symmetric cipher

We say that a cryptosystem is symmetric if the ciphering keys and the deciphering keys can be deduce from each other in polynomial time.

In practice, the key used for the ciphering will be the same than the one used for the deciphering.

☞ The inconveniency of such system is that the sender and the receiver must have the same key before starting secure transmission. Furthermore, we need a key for each user of the system.

The most known symmetric ciphers : DES, triple–DES, AES, RC4, One-time-pad.

Advantages : ciphering is very fast.

Two types of symmetric ciphers

- **Stream ciphers :** they operate on the plaintext by working on a single bit.
- **Block ciphers :** they operate by working on "large" groups of bits (e.g., 64 bits).

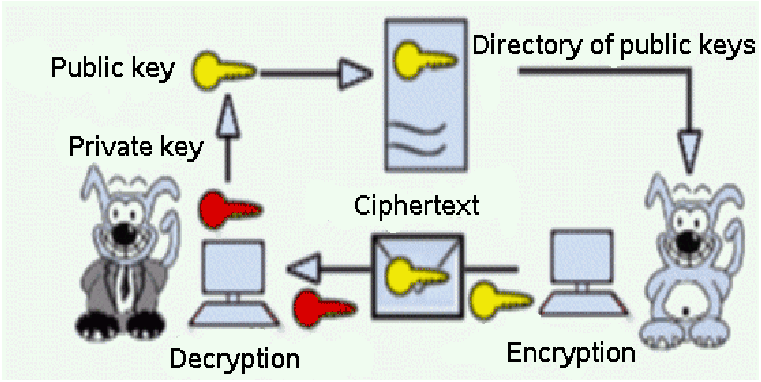
In short, stream ciphers work on the characters and block ciphers work on words.

Asymmetric cipher (a.k.a. “public key cipher”)

Such ciphers use two different keys for ciphering and deciphering. In practice, we cannot recover the deciphering key from the ciphering key in polynomial time.

☞ we can give the cipher key.

We call the ciphering key the public key of the cryptosystem and the deciphering key the private key. The last one is secret.



The principle of public key cryptography has been introduced by Whitfield Diffie and Martin Hellman in 1976 (also done separately by Ralph Merkle).

Make use of *one-way function* (**public key**) with a trapdoor (**private key**).

☞ **solve the problem of symmetric cryptography.**

Disadvantage : **100 to 1000 (even more...) times slower** than symmetric cryptography.

In practice, we mix both technics (SSL, SSH, PGP for example).

Whitfield Diffie

The fathers of public key cryptography...

Martin Hellman

Some basic notions on cryptology

We distinguish four types of attacks on the cryptographic primitives :

- **Ciphertext–only attack :** The cryptanalyst has a serie of ciphertext, eventually ciphered with the same key. The goal is to find the plaintext for a large number of ciphertext, and eventually to retrieve the key (or the keys). Mathematically : We have C_i knowing that $C_i = E_K(M_i)$ for $i = 1, .., N$, and we want to deduce M_{N+1} knowing C_{N+1}.
- **Known–plaintext attack :** The cryptanalyst has a set of plaintext with corresponding ciphertext. The problem is to retrieve the key from the data or an algorithm for the deciphering of any other ciphertext (ciphered with the same key). We know some fixed couples (M_i, C_i) with $C_i = E_K(M_i)$ for $i = 1, .., N$, and we want to deduce either M_{N+1} knowing C_{N+1}, or K. Notice that in the case of a public key scheme we can make as much couples plaintext/ciphertext that we want.

- **Choosen–plaintext or Choosen–ciphertext attack (CCA) :** The cryptanalyst can choose the couples plaintext/ciphertext. This is a dangerous attack, as the cryptanalyst can use well crafted plaintext or ciphertext (with very specific format) and then get complementary data on the key.
- **Adaptative–Choosen–plaintext or Adaptative–Choosen–ciphertext attack :** It is a special case of the previous attack where the cryptanalyst can build pairs (C_{i+1}, M_{i+1}) depending of the results at the step i.

We say that a cryptographic primitive with a given keysize has a security level of b bits if we need 2^b elementary operations in order to break the system using the best (w.r.t. running time complexity) generic attack.
Remember : Our computational limit is around 2^{80} operations (close to 2^{83} strictly speaking...). Hence, any keysize with a security level less than 80 bits is insecure !

RSA

A public key cryptosystem introduced in 1977 by Rivest, Shamir and Adleman. It is the most known and the most used.

From left to right : Leonard Adleman, Adi Shamir and Ron Rivest.

The hard problem behind RSA is the difficulty to factorise large numbers.

Example :

$2^{2^7}+1=340282366920938463463374607431768211457=5964958912749721 7\times 5704689200685129054721$.

RSA : The company

RSA®

The Security Division of EMC

Funded by the “fathers” of RSA (in fact, mainly by Ron Rivest but with shares for the others), the company became a leading worldwide company in security and cryptography (thanks also to a patent on the RSA cryptosystem, namely US patent # 4,405,829 which ended in 2001).

In 2006, RSA was bought by EMC (a leading worldwide company in information technology) for $2.1 Billions (part of Dell since 2016).

Cryptography and data security also mean big money !

RSA : the basic parameters

Private key :
p, q two primes (in practice we will impose some conditions on p and q).
d a positive integer coprime with $(p-1)\times(q-1)$.

The data which should be kept secret are p, q and d.

Public key :
$N = p \times q$
e a positive integer such as $d \times e = 1$ modulo $(p-1)\times(q-1)$.
☞ In other words, e is "the inverse of d modulo $(p-1)\times(q-1)$".

The shared public data are N and e.

We often call d the *private exponent* (or also called *secret exponent*), e the *public exponent* and N the *RSA modulus*.

Textbook RSA : Encryption/Decryption

Let M be a message (viewed, for simplicity, as a number coprime with p and q) that B wants to send to A.

- Step 1 : B get the public key (N, e) of A (via the homepage of A for instance).
- Step 2 : B computes (using modular exponentiation)

  $$C = M^e \text{ modulo } N$$

  and send the ciphertext C to A (by email for instance).
- Step 3 : A decrypt the message by computing C^d modulo N.

Key generations first problem : primality testing

Let r be an integer. We want to know if r is prime ?

Brute force method : divide by all the numbers smaller than itself

☞ highly unefficient (require r steps).

In fact we can stop dividing at $\sqrt{r}$

☞ still unefficient ($\sqrt{r}$ steps).

Fact : there exist polynomial time algorithms for testing the primality.

☞ it was proved in 2002 by Manindra Agrawal, Neeraj Kayal, and Nitin Saxena that testing the primality of an integer was a problem of type P (i.e., polynomial in time).
Their method is known as the "AKS algorithm". While it is a deterministic algorithm, current implementation are slower than previous methods.

Pseudo–primality test (Rabin/Miller)

Fermat Little Theorem :
Let p be an odd prime number and a an integer (coprime with p). Then

$$a^{p-1} \text{ modulo } p = 1.$$

Example : If $a = 3$ and $p = 17$, then

$$a^{p-1} - 1 = 3^{16} - 1 = 43046720 = 2532160 \times 17.$$

But, $2^{20} - 1 = 1048575$ is not divisible by 21. Thus 21 is not prime. A more probant example using PARI/GP. Take r a (large) random number (i.e. $\mathrm{random}(2^{500})$). Is r prime ? Assume r is odd, then compute $\mathrm{Mod}(2, r)^{(r-1)}$. Observe that it is fast ! If the result is not equal to $\mathrm{Mod}(1, r)$, then r is not prime.

Principle of the Rabin/Miller Test

- *Draw randomly an integer $a < r$.*
- *Compute a^{r-1} modulo r.*

If the result is not equal to 1, then r is not prime. Else, continue the loop in order to test "enough" a.

☞ **Very fast, but does not prove the primality.**

Remark for mathematicians : if the Riemann hypothesis is true, then we can transform the Rabin/Miller test in a deterministic primality test in $O(\log(r)^4)$.
We have (deterministic) primality test in "more or less" $O(\log(r)^6)$.

☞ **Most cryptographic standard recommend probabilistic primality tests (as Rabin/Miller) for online RSA key generations. For offline generation, deterministic primality tests can be use.**

Well choosen primes for RSA

Choosing primes for RSA modulus is not so easy and could lead to weakness.
Some standard (as in EMV) requires RSA modulus with “strong primes” p, with for instance the condition that $(p-1)/2$ is again a prime number or even stonger conditions.
The goal is to have RSA modulus which are the most robust against factorisation...
See NIST/FIPS 186-4, for details about prime generation.

The problem with RSA key generations

☞ Random numbers with bad properties or bad generation algorithms can disable security measures against potential attacks by invalidating security proofs of cryptographic algorithms or disabling countermeasures, leading to disastrous consequences.

- In 2012, Lenstra and al. (http ://eprint.iacr.org/2012/064) have analysed 11 millions of SSL/TLS certificates and found anomalous redundancies (same RSA modulus or common factors in RSA modulus) in them (around 0.004% of common factor among distinct moduli).
- Also in 2012, Heninger and al. (https ://freedom-to-tinker.com/blog/nadiah/new-research-theres-no-need-panic-over-factorable-keys-just-mind-your-ps-and-qs) have analysed similar certificates and also SSH certificates of infrastructures devices (router, firewall, access servers) and found high redundancies of RSA parameters (same RSA modulus or common factors in RSA modulus).

The problem with RSA key generations

- In a paper in Asiacrypt 2013, Bernstein, Chang, Cheng, Chou, Heninger, Lange and van Someren have shown that 184 RSA moduli can be factorized among the 2 Millions certificates from the Taiwan Citizen Digital Certificate program (RSA 1024-bit keys). This is a ratio of 0.009%, but it should be 0 ! Those certificate were issued from electronic devices (i.e., hardware based RNG=Random Number Generators).

In order to understand the previous RSA problem...

Recall that an RSA modulus N is the product of two large distinct primes p and q. Using our mathematical knowledge on the primes distribution from number theory, we get the following estimates :

n (size in bits)	estimate of the number of prime numbers with n bits	estimate of safe primes
512	$1.89063706426 \cdot 10^{151}$	$3.03068079638045 \cdot 10^{150}$
1024	$1.26691644892 \cdot 10^{305}$	$1.82777407809153 \cdot 10^{304}$
2048	$1.13851722702 \cdot 10^{613}$	$1.49321196126988 \cdot 10^{612}$

Recall that it is a wellknown fact that the number of atoms in the Universe is of the size of 10^{80} !

☞ Thus in the case of the previous experiments, the probability to get two identical primes should be ZERO from a practical viewpoint. But it is not what we observe.. !

Textbook RSA : Basic Cryptanalysis

The difficulty behind RSA is to find d knowing the public key (N, e).

☞ Factorisation of integers in polynomial time would break RSA ! (this is the *generic attack*)

But : *we do not know if breaking RSA is equivalent to factoring the modulus N ? (even if there are several evidences)*

From a practical viewpoint we need to choose the primes p and q carefully :

☞ The numbers $\frac{p-1}{2}$ and $\frac{q-1}{2}$ must not have small prime factors.

☞ The prime numbers p and q should be of the same size (i.e., more or less the same number of digits), without being too close (if not they would be too close of $\sqrt{N}$ and we could use such information to speed up the factorisation of N).

Factorisation of RSA modulus : State of the Art

Here is the current state of the art regarding the factorisation of large integers of the RSA type.

☞ The factorisation was done (in most case) using a variant of the General Number Field Sieve (GNFS).

Size (in digit) of the RSA modulus	Date
120	June, 1993
129	April, 1994
130	April, 1996
140	February, 1999
155	August, 1999
160	March, 2003
174	December, 2003
200	May, 2005
230	December, 2009.

RSA768 (more or less 230 digits) was factorised the 12 December 2009. Here are the prime numbers p and q for the RSA modulus N with 200 digits

$p = 3532461934402770121272604978198464368671197400197625023649303468776121253679423200058547956528088349$

$q = 7925869954478333033347085841480059687737975857364219960734330341455767872818152135381409304740185467$

RSA and the factorisation of the modulus

We can summarise the relationship between RSA and the factorisation with the following statement :

*From the factorisation of N we can retrieve $\varphi(N) = (p-1)(q-1)$, and then get d. On the other hand, the knowledge of $\varphi(N)$ or d give the factorisation of N (**in polynomial time**).*

We will prove this result. The main consequence is the following fact : *if we know a polynomial–time algorithm for finding d, we have a polynomial–time algorithm for the factorisation of N*.

RSA : complexity of best generic attacks

Algorithms	**Complexity**
Rho method (Pollard/Brent, 1975-80)	$O(\sqrt{p})$
ECM (H. Lenstra, 1987)	$O(\exp((\sqrt{2}+o(1))\sqrt{\log(p)\log(\log(p))}))$
(MP)QS (Pomerance, 1984-1990)	$O(\exp((1+o(1))\sqrt{\log(N)\log(\log(N))}))$
(G)NFS (Pollard and al, 1988-present)	$O(\exp((C+o(1))\log(N)^{\frac{1}{3}}\log(\log(N))^{\frac{2}{3}}))$, $C=(64/9)^{\frac{1}{3}}$

In red exponential complexity and in blue subexponential complexity. p is the largest prime factor. The shapes of $p-1$ and $q-1$ play a crucial role in the factorisation algorithms.

RSA : bits security

bits of security	**RSA key size**
80	1024
112	2048
128	3072
256	15360

If we have n bits of security, it means that we need $O(2^n)$ operations in order to recover the private key.

Quid de la taille de mes mots de passe ?

Le niveau de sécurité usuel est de 128 bits, une clef RSA 2048 bits (usuellement utilisée pour un certificat SSL pour une connexion web sécurisée) à une sécurité de 112 bits.
Sous l'hypothèse que le chiffrement du mot de passe se fait en «temps rapide» et que l'on utilise 80 caractères alphanumériques et spéciaux pour sa construction, nous avons la table de correspondance suivante :

bits de sécurité	**taille du mot de passe (nb de carac.)**
80	13
112	18
128	21
256	41

Conclusion : si vous utilisez moins de 13 caractères quoi qu'il arrive, alors il est préférable d'attaquer votre portefeuille électronique que de casser vos clefs cryptographiques !

Petite démo pour se faire peur… !

PARI/GP : The reference version

For experiment with public key ciphers, we will use the software PARI/GP. It is a software specialised for number theory (including elliptic curves) and it is easy to use, with a simple C-like programming language.

`http://pari.math.u-bordeaux.fr`

☞ We will use the PARI version 2.7.4 in 64bits or the 2.8.0 dev version

Why we cannot use Textbook RSA ?

- Determinism of the primitive (dictionary attack).
- Vulnerable to CCA.
- Vulnerable to broadcast attacks.
- ...

Problem with textbook RSA (part I) : broadcast attack

Suppose that A, B and C have a common public exponent e of small size, say for instance $e = 3$, but with distinct RSA moduli N_i for $i = A, B, C$ that we can assume coprimes.
Suppose that X send the same message M to A, B and C (this is a real–life scenario, a big company can send internal data to its offices worldwide). We intercept $M^3 \mod N_i$ for $i = A, B, C$.
Thus we have $m = M^3 \mod N_A N_B N_C$. But M is smaller than each of the N_i (if not we loose information).
So, we can deduce that M^3 is smaller than the product of the N_i.
Conclusion, $m = M^3$. Then, it is enough to extract the cubic root of m to retrieve M.

☞ *We did not factorise one of the RSA modules and we did not recover the private keys. But with this protocol, we have built an alternative decryption algorithm. It is an example of global deduction.*

Illustration of this attack

Take $N_A = 3208514389$, $N_B = 3209546489$ and $N_C = 3210688883$. Suppose that $M = 2330592019$. We then have the following equalities :

$$\begin{aligned} M^3 \mod N_A &= 1590232416\,, \\ M^3 \mod N_B &= 2258380306\,, \\ M^3 \mod N_C &= 719706713\,. \end{aligned}$$

By the chinese remainder theorem (CRT), we get

$$M^3 \mod N_A N_B N_C = 12658981485955404228479142859\,.$$

But $\sqrt[3]{12658981485955404228479142859} = 2330592019.$
We got M ! Notice that the computations are fast. We will see a generalisation of this attack later.

Problem with Textbook RSA (part II) : Insecure against CCA

Suppose that (N, e, d) is a set of RSA parameters. Let C be a ciphertext which corresponds to the plaintext M. Now, suppose that as an attacker, we have the possibility to perform a choosen-ciphertext attack ; we choose randomly an integer r with $1 < r < N$ and such that $\gcd(r, N) = 1$ (which is the most likely !). We set $C' = r^e C \mod N$ and ask for the corresponding plaintext. Then we recover

$$C'^d \mod N = r^{ed} C^d \mod N = rM \mod N\,.$$

But as r is invertible modulo N, we can retrieve M from $C'^d \mod N$.

RSA : Improving the computing cost of decryption and signature

We have seen that decryption time for RSA is costly. An idea is to compute modulo p and q instead of computing modulo $N = p \times q$. Suppose that N is our RSA modulus, d our private exponent and e our public exponent. We suppose that we get a ciphertext $C = M^e \mod N$ (recall that $a \mod b$ means that we are working with the remainder of the euclidean division of a by b and performing modular operations). Set $d_p = d \mod (p-1)$ and $d_q = d \mod (q-1)$. In order to decrypt C, we compute the following

$$\begin{cases} M_p = C^{d_p} \mod p\,, \\ M_q = C^{d_q} \mod q\,. \end{cases}$$

Then we apply the CRT (Chinese Remainder Theorem) on M_p (w.r.t. p) and M_q (w.r.t. q) in order to retrieve M (w.r.t. $N = p \times q$). As modular exponentiation can be done in $O(\log(N)^3)$ and as roughly p and q are half the size of N, we should expect a gain speed of at most 400% in comparison to a classical decryption.

RSA : Comparison of decription costs

The following table give timings for 1000 decryptions with respect to an RSA modulus of size n with or without CRT.

n	without CRT (in ms)	with CRT (in ms)	speed factor
1024	2753	812	3.39
2048	17261	4901	3.52
4096	119084	33735	3.53

Timings with PARI/GP 2.4.3 on Xeon5160 at 3Ghz, Linux 64 bits.

☞ We notice that, *in such arithmetic*, multiplying by 2 the size of the RSA modulus implies a multiplication by 6 of decryption time.

RSA : An example of weakness introduced by the use of CRT in signature

Let $N = pq$ be an RSA modulus with public exponent e and private exponent d. We set $d_p = d \mod (p-1)$ and $d_q = d \mod (q-1)$. Let M be a message whose signature is

$$s = M^d \mod N\,.$$

The signature of the message is verified by computing $M' = s^e \mod N$ and then checking that $M = M'$.
In order to accelerate the signing operations, the signer computes

$$s_p = m^{d_p} \mod p \qquad \text{and} \qquad s_q = m^{d_q} \mod q\,.$$

The signature s can be computed as $s = as_p + bs_q \mod N$ where a and b are integers satisfying

$$a = \begin{cases} 1 & \mod p \\ 0 & \mod q \end{cases} \qquad \text{and} \qquad b = \begin{cases} 0 & \mod p \\ 1 & \mod q\,. \end{cases}$$

Fault analysis on the RSA signature scheme with CRT

Suppose that an error occurs during the computation of s_p (i.e., $s_p \neq M^{d_p} \mod p$) and no errors occur during the computation of s_q (i.e., $s_q = M^{d_q} \mod q$). Then an adversary who obtains the message M and the (incorrect) signature s can easily factor N (i.e., in linear time) and thereafter compute the private exponent d.

Indeed, we can compute $\gcd(s^e - M, N)$. Due to the error, the gcd is non trivial and will reveal either p or q.

Modified RSA version (allowed in many standard, e.g. IEEE, PKCS)

For $N = pq$, where p and q are distinct odd primes, define

$$\lambda(N) = \frac{(p-1)(q-1)}{\gcd(p-1, q-1)}\,.$$

We require that $ed = 1 \mod \lambda(N)$ (instead of $ed = 1 \mod \varphi(N)$). Then RSA encryption and decryption sill work (exercise).

The Wiener's attack against RSA

This attack will emphasise the fact that small private exponents are dangerous. The method of Michael J. Wiener (1989) is based on a mathematical tool called "continued fractions". Historically, continued fractions were introduced for computing rational approximation of irrational numbers (and more generally real numbers).

Wiener's result on low private exponent

Wiener's Theorem (1989) : *Let $N = pq$ be an RSA modulus with $q < p < 2q$. Suppose that $d < \frac{1}{3}N^{\frac{1}{4}}$. Then, given (N, e) with $ed = 1 \mod \varphi(N)$, we can recover d in linear time (i.e., in $O(\log_2(N))$).*

Remark : from practical viewpoint, the attack still works if the size of the private exponent is below half the size of the modulus.

Real life RSA protocols

RSA Encryption schemes

An encryption scheme consists of an encryption operation and a decryption operation, where the encryption operation produces a ciphertext from a message with a recipient's RSA public key, and the decryption operation recovers the message from the ciphertext with the recipient's corresponding RSA private key.

In the remaining text, we will use the following notations : the symbol $||$ denote the concatenation of strings, $\oplus$ the bitwise XOR and $|\cdot|$ the (string) binary length.

Examples :

$101||001 = 101001$, $101||01 = 10101$, $|101| = 3$, $|01| = 2$, $|001| = 3$.

$101 \oplus 001 = 100$, $11001 \oplus 00110 = 11111$.

RSA–OAEP

RSA -OAEP (RSA Encryption Scheme - Optimal Asymmetric Encryption Padding) is a public-key encryption scheme combining the RSA algorithm with the OAEP method. The inventors of OAEP are Mihir Bellare and Phillip Rogaway, with enhancements by Don B. Johnson and Stephen M. Matyas.
The OAEP method can be used with general asymmetric algorithm.

General description of the OAEP scheme

Before applying an encryption algorithm such as RSA, preprocessing of the message is necessary in order to prevent known attacks (in order also to break some mathematical structures inherent to the asymmetric cipher, such as the multiplicative structure in the case of RSA, and which could be useful for the attacker).
The message is divided into blocks and then some formatting and/or padding mechanisms are performed. The OAEP mechanism can be used with any bijective trapdoor function.
The RSA-OAEP and RSA-PSS protocols are fully described in PKCS #1.

The settings of the OAEP

- A trapdoor function $f : D \to D$ with $D \subset \{0,1\}^n$ for a given n,
- A deterministic pseudorandom bit generator $G : \{0,1\}^k \to \{0,1\}^l$ with $n = k + l$,
- A (cryptographic) hash function $h : \{0,1\}^l \to \{0,1\}^k$ (assuming in general that $k < l$).

Given a random seed $s \in \{0,1\}^k$, G generates a pseudorandom bit sequence of length l.

OAEP Encryption

To encrypt a message $m \in \{0,1\}^l$, we proceed in three steps :

1. We choose a random bit string $r \in \{0,1\}^k$,
2. We set $x = (m \oplus G(r)) || (r \oplus h(m \oplus G(r)))$. If $x \notin D$ we return to step 1.
3. We compute $c = f(x)$.

Comments on OAEP Encryption

We can see that $x = a||b$ with $|a| = l$ and $|b| = k$. Hence the message is in the first l bits of x masked with $G(r)$ while the seed r is in the last k bits of x masked by the hash of its first l bits.
We should also notice, that G is implicitly supposed to be computationally easy as well as the hash function. In practice, they are often computed in linear time. Hence, the most time-consuming is the function f, as it should be.
In other words, *the formating/masking/padding should not increase significantly the encryption time*.

Of course, thanks to the randomness, the encryption is no longer deterministic. Encrypting the same message twice will produce different ciphertexts.

OAEP Decryption

To decrypt a ciphertext c, we use the function f^{-1}, the same determinsitic pseudorandom bit generator G and the same h as above :

1. Compute $f^{-1}(c) = a||b$ with $|a| = l$ and $|b| = k$.
2. Set $r = h(a) \oplus b$ and get $m = a \oplus G(r)$.

Remark : A good secure hash function can be used for constructing G. In fact, we can use hash functions for building cryptographic PRNG. It is the size of the hash function which will give the size of k.

Exercises : Check the validity of RSA AOEP (i.e., Encryption/Decryption works).

Comments on OAEP

To compute the plaintext m from the ciphertext c, an adversary must figure out all the bits of x from $c = f(x)$. He/She needs the first l bits to compute $h(a)$ and the last k bits to get r. Therefore, an adversary cannot exploit any advantage from some partial knowledge of x. Furthremore, we see that the multiplicative structure of RSA is no longer preserved and it is probabilistic. The low exponent encryption attack against RSA (with broadcasting) is also prevented, provided that the plaintext is individually re-encoded by OAEP for each recipient before it is encrypted with RSA.

A slight modification of the basic scheme is the following : Let k, l as before and let k' be another parameter with $n = l + k + k'$. We use a pseudorandom generator $G : \{0,1\}^k \to \{0,1\}^{l+k'}$ and a cryptographic hash function $h : \{0,1\}^{l+k'} \to \{0,1\}^k$. To encrypt a message $m \in \{0,1\}^l$, we first append k' 0-bits to m and then we encrypt the extended message as before. This modified scheme is proved semanticaly secure (mainly secure against certain type of chosen-ciphertext attacks).

RSA Signature schemes

RSASSA-PSS (RSA Signature Scheme with Appendix - Probabilistic Signature Scheme) is an asymmetric signature scheme with appendix combining the RSA algorithm with the PSS encoding method. The inventors of the PSS encoding method are Mihir Bellare and Phillip Rogaway. This is very close in spirit to RSA–OAEP. As usual you will see the technical appendix for a description of the scheme according to usual standards. Let describe here the general framework.

The signature of a message depends on the message and some randomly choosen input. The resulting signature scheme is therefore probabilistic. Let $f : D \to D\ D \subset \{0,1\}^n$ for a given n, be a trapdoor bijective one-way function,
$G : \{0,1\}^l \to \{0,1\}^k \times \{0,1\}^{n-(l+k)}$ with $n > k + l$,
$x \mapsto (G_1(x), G_2(x))$ and a (cryptographic) hash function
$h : \{0,1\}^* \to \{0,1\}^l$. The PSS is applicable to messages of arbitrary length. The message m cannot be recovered from the signature σ.

Signing

To sign a message $m \in \{0,1\}^*$, we proceed in three steps :

1. We choose $r \in \{0,1\}^k$ at random and compute $x = h(m||r)$,
2. We compute $G(x) = (G_1(x), G_2(x))$ and $y = x||(G_1(x) \oplus r)||G_2(x)$. If $y \notin D$ we return to step 1.
3. The signature of m is $\sigma = f^{-1}(y)$.

We notice that all bits of y depend of the message m. The hope is that the mapping $m \mapsto y$ behaves like a truly random function.

Verification

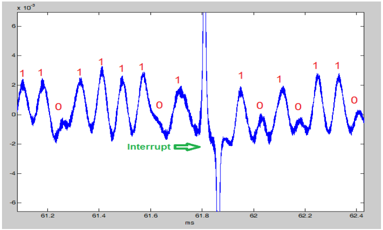
To verify the signature of a signed message (m, σ), we use the same parameters than above and proceed as follows :

1. Compute $f(\sigma)$ and decompose $f(\sigma) = u||v||w$ where $|u| = l$, $|v| = k$ and $|w| = n - (k + l)$.
2. Compute $r = v \oplus G_1(u)$.
3. We accept the signature σ if $h(m||r) = u$ and $G_2(u) = w$.

Remark : the goal of G_2 is to provide an integrity check.

Physical attack on RSA

Genkin, Pipman and Tromer demonstrated (2014) physical side-channel attacks on a popular software implementation of RSA, running on laptop computers. Our attacks use novel side channels and are based on the observation that the "ground" electric potential in many computers fluctuates in a computation-dependent way. Similar attack was known, but on hardware implementation.



(C) Genkin, Pipman and Tromer / Tel Aviv Univ. 2014.

☞ Necessary to protect software implementation of cryptographic primitives...

Rijndael (aka AES) : The design of an “unconventional” block cipher

complementary references for the AES part

- The NIST AES pages (archived) : `http://csrc.nist.gov/archive/aes/index.html`
- The FIPS standard 197 : `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`
- *The Design of RijndaeL : AES - The Advanced Encryption Standard* by Joan Daemen and Vincent Rijmen (Springer, 2002)
- Chapter 4 : The Advanced Encryption Standard (AES) of the book *Understanding Cryptography* by Paar and Pelzl `http://wiki.crypto.rub.de/Buch/download/Understanding-Cryptography-Chapter4.pdf`

Short overview on block ciphers

Basic block ciphers design (Part I)

We will denote by V_k the space of k-bits vectors with k a positive integer and $\mathcal{K}$ a subset of V_k that we will call the **keyspace**.

Definition of block cipher : *An n-bits block cipher is a map* $E : V_n \times \mathcal{K} \to V_n$ *such that for each key* $K \in \mathcal{K}$, $E(-, K) : V_n \to V_n$ *is a bijection. We will denote* E_K *this map and* D_K *its inverse. The map* E_K *will be called the encryption map and* D_K *the decryption map.*

Basic block ciphers design (Part II)

we say that a block cipher is an *iterated block cipher* if it is built as an iteration of internal functions called *round function*. Each step using the round function is called *a round*. The parameters of such system are : the number of rounds r, the size (in bits) n of the blocks and the size k of the key K. From K we derive r *subkeys* K_i (also called *round keys*). For each K_i, the round function is a bijection on the entries of the round. The derivation of the K_i from K is called a *key shedule*.

Feistel network

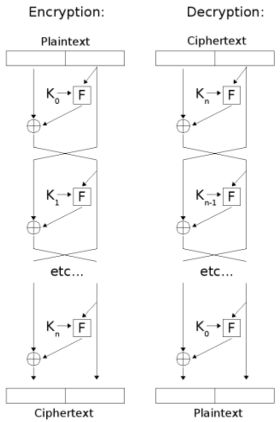
A *Feistel cipher* (a.k.a. *Feistel network* or *Feistel scheme*) is an iterated block cipher which transform a plaintext of $2t$-bits (L_0, R_0), with L_0 and R_0 of t bits, in a ciphertext (L_r, R_r), using r rounds ($r \geqslant 1$), such that the ith round transforms (L_{i-1}, R_{i-1}) in (L_i, R_i) by using the subkey K_i as follows :

$$L_i = R_{i-1}\,, \quad R_i = L_{i-1} \oplus f(R_{i-1}, K_i)\,,$$

with K_i derived from K. The function f being *eventually* arbitrary.

Horst Feistel (1915–1990)
(source and copyright : IBM)

Example of Feistel scheme



Feistel Cipher

DES, Triple DES, Blowfish, Serpent, KASUMI, Twofish are examples of block ciphers based on Feistel schemes.

Comments on the Feistel scheme

- **Purpose** : get a bijective function from a general f.
- **Disadvantage** : one gets a part of the input in the output ; so we need several rounds.
- In the opposite direction, we have :

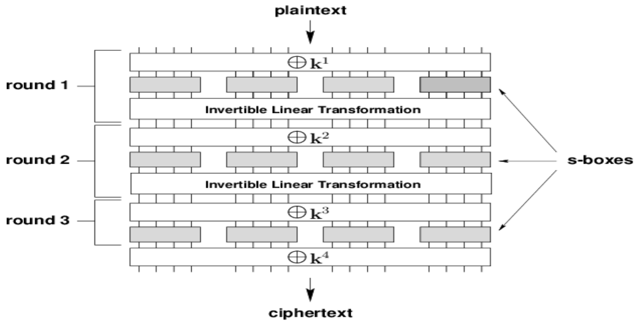
$$R_{i-1} = L_i, \quad L_{i-1} = R_i \oplus f(L_i, K_i)$$

So it is enough to change the direction of time by replacing i by $r - i$ (with r the number of rounds) and swapping L_i and R_i for every $i \geqslant 0$.

SPN : Substitution–Permutation Network

It is a combination of simple transformations, mainly made of what we call *S*-box (a.k.a. "Substitution box") and *P*-box (a.k.a. "Permutation box") in order to construct more robust cryptosystems.

The following picture illustrates this type of scheme



SPN in action

(source : paper of Keliher/Meijer/Talares)

SPN : Understanding Permutation and Substitution (Part 1)

In Mathematics a permutation is also called a substitution.

☞ **But in Cryptology the two words have different meaning :**

A *permutation* acts on the *position* of the entries while a *substitution* acts on their *values*.

Definition : *A* permutation *is a bijection from the set* $\{1, ..., n\}$ *to itself.*

More generally, a permutation is a bijection from a finite set to itself. But any finite set with n elements is in bijection with $\{1, ..., n\}$.

We note $\mathfrak{S}_n$ the set of permutations of $\{1, ..., n\}$ with the composition (of functions). From the mathematical viewpoint, $\mathfrak{S}_n$ is a group with neutral element given by the identity map. If σ is an element of $\mathfrak{S}_n$, we will represent it as

$$\sigma = \begin{pmatrix} 1 & 2 & \cdots & n \\ \sigma(1) & \sigma(2) & \cdots & \sigma(n) \end{pmatrix}.$$

SPN : Understanding Permutation and Substitution (Part 2)

As a simple example with

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 4 & 1 & 5 & 2 & 6 & 3 & 7 \end{pmatrix},$$

we get the transformation (as *permutation*)

$$\begin{array}{ccccccc} \mathsf{W} & \mathsf{E} & \mathsf{L} & \mathsf{C} & \mathsf{O} & \mathsf{M} & \mathsf{E} \\ & & & \downarrow & & & \\ \mathsf{C} & \mathsf{W} & \mathsf{O} & \mathsf{E} & \mathsf{M} & \mathsf{L} & \mathsf{E} \end{array}$$

SPN : Understanding Permutation and Substitution (Part 3)

Consider again

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 4 & 1 & 5 & 2 & 6 & 3 & 7 \end{pmatrix},$$

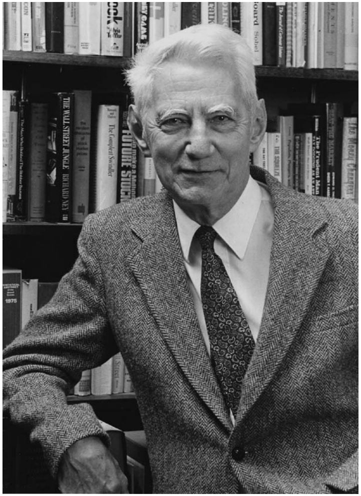
Now consider the message $M = 2164523$. If we encrypt the message M using the *substitution* π, we get

$$1432615 = \pi(2)\pi(1)\pi(6)\pi(4)\pi(5)\pi(2)\pi(3)\,.$$

If we encrypt M using the *permutation* π, we get

$$4251263\,.$$

Confusion/Diffusion from Shannon



Claude Shannon (1916-2001).

Picture from Lucent technologies ®

Claude Shannon is the father of information theory, including several key concepts of coding theory and cryptography. One of its key result is the “Shannon theorem” for transmission’s channel. But he is also the father of the concepts of *confusion* and *diffusion* used in the design of symmetric ciphers.

Confusion : avoid relations between the plaintext and the ciphertext (e.g., avoid linearity). Tool : *Substitutions boxes*.

Diffusion : spread the information in each bit of the plaintext in the ciphertext. Tool : *Permutations boxes*.

DES as a fundamental example of Feistel scheme

DES (Data Encryption Standard) is a Feistel cipher with $n = 64$ bits, producing ciphertext of 64 bits. The size of the key is 56 bits with 8 supplementary bits used for parity tests. It has been designed by IBM (Cryptographic Lab including Horst Feistel and Don Coppersmith among others) at the beginning of the 70's as a call from the NIST (formerly National Bureau of Standard) and with "involvement" from the NSA.
The cipher is made of 16 rounds with 16 subkeys of 48 bits. The function f is built from S-boxes and P-boxes.

☞ It was used for encryption as well as for decryption on easily implanted operations in hardware. It can reach a speed of encryption > 300Mb/s with specific hardware existing in the early 90's (of course we could reach > 10Gb/s today).

DES : Short description

If L_{i-1} and R_{i-1} are the half-block of 32 bits produced at the step $i-1$, we build L_i and R_i as follows :

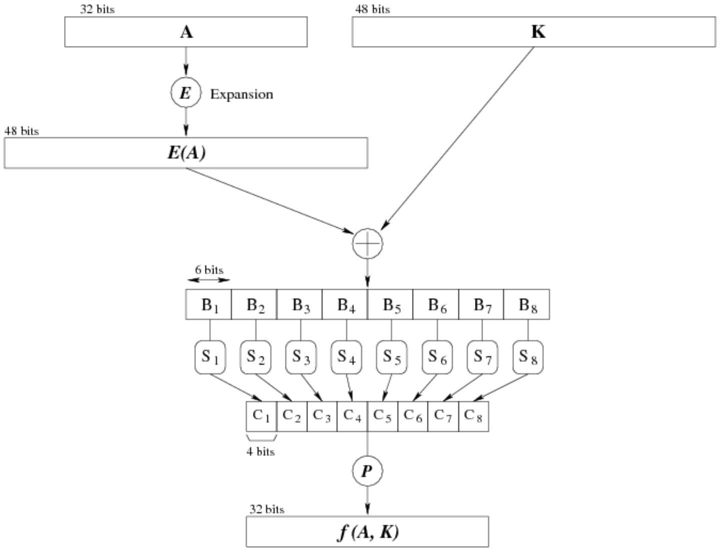
$$\begin{aligned} L_i &= R_{i-1}\,, \\ R_i &= L_{i-1} \oplus f(R_{i-1}, K_i)\,, \\ f(R_{i-1}, K_i) &= P(S(E(R_{i-1}) \oplus K_i))\,, \end{aligned}$$

where E is the expansion function tranforming R_{i-1} in a word of 48 bits. The function P is a permutation of 32 bits (P-box) and S is an S-box. The number of steps (a.k.a., rounds) is 16.

In order to add some diffusion, we perform an Initial Permutation (often called IP) on the 64 bits input and a Final Permutation on the 64 bits output. This final permutation is given as the inverse of the initial permutation (i.e., it is IP^{-1}).

☞ *In the DES, the Feistel function f is a complicated function designed in order to mask the subkeys.*

DES : Feistel function General Chart



AES : Advanced Encryption Standard

While known today as the name of a symmetric cipher, it is in fact the result of a selective process done by the NIST. It became a NIST standard in 2001.
In the following we will see ; the selection process of AES, the motivations for AES, the design of Rijndael (aka AES) ; strength and weakeness of AES.

The AES Selection Process in a nutshell

- NIST call for algorithms : 12 september 1997,
- Candidates : 15 kept on 20 august 1998,
- Finalists : MARS, RC6, Rijndael, Serpent and Twofish,
- Final choice (2 october 2000) : Rijndael designed by two belgians Joan Daemen and Vincent Rijmen ; pronounced "Reign Dahl", "Rain Doll", "Rhine Dahl"....
- FIPS197 (26 november 2001).

Joan Daemen and Vincent Rijmen (source : Darwin Mag.)

Why a new standard ?

- DES has become attackable by exhaustive search,
- The natural improvements lead to rather inefficient implementations,
- New capabilities now available in highspeed enciphering : use processor instructions,
- Development of the evaluation of systems : differential and linear analysis.

Why a worldwide public call ?

- Gather work from the cryptographic community ; few specialists as yet in academic research,
- Stimulate research into secure systems,
- Prevent backdoors,
- Accelerate acceptance and adoption of the standard.

NIST : the final words on Rijndael

In the conclusion of the final report ("Report on the developement of the AES", 2000), NIST motivates the choice of Rijndael with the final words :

Rijndael appears to be consistenly a very good performer in both hardware and software accros a wide range of computing environments regardless of its use in feedback or non-feedback modes. Its key setup is excellent, and its key agility is good. Rijndael's very low memory requirements make it very well suited for restricted space environments, in which it also demonstrates excellent performance. Rijndael's operations are among the easiest to defend against power and timing attacks. Additionally, it appears that some defense can be provided against such attacks without significantly impacting Rijndael's performance. Finally, Rijndael's internal round structure appears to have good potential to benefit from instruction-level parallelism.

Rijndael(AES) : Overall description

We work with the field $\mathbb{F}_{256}$, where we represent elements (1 byte of storage) as two hexadecimal numbers (corresponding to the bits representation as $\mathbb{F}_2$ vector space).
The system uses a number of $r =$10, 12 or 14 rounds, each including 4 operations on the block (plus a last one which only has 3).
The data block is represented by a matrix of bytes having 4 lines and a number of columns Nb that is 4, 6 or 8. Therefore, the block contains $32 \times Nb$ bits :

a_{00}	a_{01}	a_{02}	a_{03}	a_{04}	a_{05}
a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}
a_{20}	a_{21}	a_{22}	a_{23}	a_{24}	a_{25}
a_{30}	a_{31}	a_{32}	a_{33}	a_{34}	a_{35}

Example of data block for $Nb = 6$.

Similarly, the key is represented by a matrix of 4 lines and Nk columns, where Nk =4, 6 or 8, regardless of Nb :

k_{00}	k_{01}	k_{02}	k_{03}
k_{10}	k_{11}	k_{12}	k_{13}
k_{20}	k_{21}	k_{22}	k_{23}
k_{30}	k_{31}	k_{32}	k_{33}

The number of rounds r is set according to the values of Nb and Nk, according to the table :

$Nk \backslash Nb$	4	6	8
4	10	12	14
6	12	12	14
8	14	14	14

Round Operations

Rijndael starts with an XOR, with the index subkey 0. Then r rounds.
The operations within a round can be described as follows :

- SubBytes : An affine map on the vector space of dimension 8 on $\mathbb{F}_2$ operating on the **inverse** of the string.
- ShiftRows : offset lines after representations of the block in the rectangular array
- MixColumns : Multiplication modulo $X^4 + 1$ by $[03]X^3 + [01]X^2 + [01] + [02]$ where the column vectors are translated in polynomials over the field $\mathbb{F}_{256}$.
- AddRoundKey (or RoundKey Addition) : XOR by a subkey for the round number.

The first $r - 1$ rounds perform these 4 operations in the stated order. The final round does not perform a MixColumns.

SubBytes (1/5) : Design criteria

1. **Non-linearity** : those (sub)criteria were inspired by linear and differential cryptanalysis.
   1. **Correlation** : the maximum input/output correlation amplitude must be as small as possible
   2. **Difference propagation probability** : the maximum difference propagation probability must be as small as possible.
2. **Algebraic complexity** : the algebraic expression of the S-box in $\mathbb{F}_{2^8}$ has to be complex.

SubBytes (2/5) : $\mathbb{F}_{256}$

This is the field generated on $\mathbb{F}_2$ by a root x of the irreducible polynomial.

$$X^8 + X^4 + X^3 + X + 1$$

An element is written as a polynomial of degree 7 or less with coefficients 0 or 1. For instance

$$x^6 + x^4 + x^2 + x + 1$$

is represented by the byte : 01010111= [57] (hexadecimal)

☞ But, you can choose another presentation of the field. Either with an other irreducible polynomial or see $\mathbb{F}_{256}$ as, either, an extension of $\mathbb{F}_4$ or $\mathbb{F}_{16}$. In such case, you will have to redesign the different operations of AES. By doing so, you can either reduce the memory needed for AES (at the cost of speed) or increase the speed of AES (often at the cost of memory).

SubBytes (3/5)

SubBytes starts by replacing x with y, which is its inverse in $\mathbb{F}_{256}$ or 0 if $x = 0$.
Next, by interpreting the elements of $\mathbb{F}_{256}$ as 8 column matrices :

$$\begin{pmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \\ z_6 \\ z_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

SubBytes (4/5) : table written with hexadecimal digits

63	7c	77	7b	f2	6b	6f	c5	30	1	67	2b	fe	d7	ab	76
ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
4	c7	23	c3	18	96	5	9a	7	12	80	e2	eb	27	b2	75
9	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
53	d1	0	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
d0	ef	aa	fb	43	4d	33	85	45	f9	2	7f	50	3c	9f	a8
51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
cd	c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	b	db
e0	32	3a	a	49	6	24	5c	c2	d3	ac	62	91	95	e4	79
e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	8
ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
70	3e	b5	66	48	3	f6	e	61	35	57	b9	86	c1	1d	9e
e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
8c	a1	89	d	bf	e6	42	68	41	99	2d	f	b0	54	bb	16

SubBytes (5/5) : table written with decimal digits

Written with decimal digits the table is wider :

99	124	119	123	242	107	111	197	48	1	103	43	254	215	171	118
202	130	201	125	250	89	71	240	173	212	162	175	156	164	114	192
183	253	147	38	54	63	247	204	52	165	229	241	113	216	49	21
4	199	35	195	24	150	5	154	7	18	128	226	235	39	178	117
9	131	44	26	27	110	90	160	82	59	214	179	41	227	47	132
83	209	0	237	32	252	177	91	106	203	190	57	74	76	88	207
208	239	170	251	67	77	51	133	69	249	2	127	80	60	159	168
81	163	64	143	146	157	56	245	188	182	218	33	16	255	243	210
205	12	19	236	95	151	68	23	196	167	126	61	100	93	25	115
96	129	79	220	34	42	144	136	70	238	184	20	222	94	11	219
224	50	58	10	73	6	36	92	194	211	172	98	145	149	228	121
231	200	55	109	141	213	78	169	108	86	244	234	101	122	174	8
186	120	37	46	28	166	180	198	232	221	116	31	75	189	139	138
112	62	181	102	72	3	246	14	97	53	87	185	134	193	29	158
225	248	152	17	105	217	142	148	155	30	135	233	206	85	40	223
140	161	137	13	191	230	66	104	65	153	45	15	176	84	187	22

ShiftRows (1/2)

The ShiftRows step is a byte transposition that cyclically shifts the rows of the state over (four) different offsets.

Design criteria for the offsets

- **Diffusion optimal** : the four offsets have have to be different.
- **Other diffusion effects** : the resistance against truncated differential attacks and saturation attacks has to be maximised.

ShiftRows (2/2)

One offsets line *i* by *i* positions to the left in the array :

a_{00}	a_{01}	a_{02}	a_{03}
a_{10}	a_{11}	a_{12}	a_{13}
a_{20}	a_{21}	a_{22}	a_{23}
a_{30}	a_{31}	a_{32}	a_{33}

$\rightarrow$

a_{00}	a_{01}	a_{02}	a_{03}
a_{11}	a_{12}	a_{13}	a_{10}
a_{22}	a_{23}	a_{20}	a_{21}
a_{33}	a_{30}	a_{31}	a_{32}

MixColumns (1/2)

The MixColumns step is a bricklayer permutation operating on the state column by column.

Design criteria :

- **Dimensions** : the transformation is a bricklayer transformation operating on 4-byte columns (optimal for 32-bit architecture using look-up table).
- **Linearity** : the transformation is preferably linear over $\mathbb{F}_2$ (imposed by the internal architecture of the cipher)
- **Diffusion** : the transformation has to have *relevant* diffusion power (imposed by the internal architecture of the cipher).
- **Performance on 8-bit processors** : the performance of the transformation on 8-bit processors has to be high (because for this step, performance on 8-bit processors is difficult to obtain for).

MixColumns (2/2)

We convert each column of the data matrix. A column with 4 entries in $\mathbb{F}_{256}$ corresponds to a polynomial of degree 3. You multiply it by $C(X) = [03]X^3 + [01]X^2 + [01]X + [02]$ and you take the remainder of the division by $X^4 + 1 \in \mathbb{F}_{256}[X]$ From a matrix viewpoint, this corresponds to :

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}$$

Notice that this transformation is invertible, the inverse corresponds to the polynomial $D(X) = [0B]X^3 + [0D]X^2 + [09]X + [0E]$

Key Schedule (1/5)

Design Criteria

The conceptors of Rijndael have designed the key expansion according to the following criteria

- **Efficiency** : It should be possible to execute the key schedule using a small amount of working memory and to have a high performance on a wide range of processors.
- **Symmetry elimination** : It should use round constants to eliminate symmetries.
- **Diffusion** : It should have an efficient diffusion of cipher key differences into the expanded key.
- **Non-linearity** : It should exhibit enough non-linearity to prohibit the full determination of diffenrences in the expanded key from cipher key differences only.

Key Schedule (2/5)

Beginning : a key of $4\times Nk$ bytes. We want $r+1$ keys of $4\times Nb$ bytes each round.
You can then perform an XOR coefficient by coefficient.
We start with developing the key into an array $W[i]$ of $Nb\times(r+1)$ entries, each one is 4 bytes long. The subkey relative to the number i round ($i=0,\cdots,r$) is simply composed of the part of the array $W[j]$ with $Nb\times i\leqslant j<Nb\times(i+1)$.

Key Schedule (3/5)

Development of the key if $Nk \leqslant 6$: You use the recursive program below :

```
for $\{i = 0; i < Nk; i++\}$
    $W[i] = (key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])$ ;
for $\{i = 4; i < Nb*(r+1); i++\}$
    $\{temp = W[i-1]$ ;
    if ($i$%Nk==0)
        $temp = SubByte(RotByte(temp)) \oplus Rcon[i/Nk]; \}$
    $W[i] = W[i-Nk] \oplus temp$ ;
    $\}$
$\}$
```

starting with the key to fill W for the indexes $j < Nk$.
SubByte as above. *RotByte* : $(a, b, c, d) \to (b, c, d, a)$ Rcon is here in order to break symmetries (in order to avoid “weak keys”).

Key Schedule (4/5)

For $Nk > 6$ (i.e., $Nk = 8$) : we also apply *SubBytes* before the final $\oplus$.
If Nk divides $i - 4$; here is how :

```
for $\{i = 0; i < Nk; i++\}$
    $W[i] = (key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])$ ;
for $\{i = 4; i < Nb*(r+1); i++\}$
    $\{temp = W[i-1]$ ;
    if $(i\%Nk{==}0)$
        $temp = SubByte(RotByte(temp)) \oplus Rcon[i/Nk];\}$
    else if $(i\%Nk{==}4)$
        $temp = SubByte(temp)$ ;
    $W[i] = W[i-Nk] \oplus temp$ ;
    }
}
```

Key Schedule (5/5)

We needed constants
$Rcon[i] = (RC[i], 00, 00, 00)$
where $RC[i]$ is the binary representation of x^i,
here $x = [02]$ is the class of X.
The Rcon data is here in order to avoid symmetries.
☞ the deciphering is done with an algorithm that has the same structure. It is just that the polynomials chosen are different and, according to the designers, give rise to slower calculations.

Demo time...

Let us illustrate step-by-step the Rijndael scheme using the Rijndael tools made by Enrique Zabala

`http://www.formaestudio.com/rijndaelinspector/`

Appendix on $\mathbb{F}_{256}$

Here we give the list of the irreducible polynomials of degree 8 over $\mathbb{F}_2$:

$$
\begin{gathered}
x^8+x^4+x^3+x+1, \quad x^8+x^4+x^3+x^2+1 \\
x^8+x^5+x^3+x+1, \quad x^8+x^5+x^3+x^2+1 \\
x^8+x^5+x^4+x^3+1, \quad x^8+x^5+x^4+x^3+x^2+x+1 \\
x^8+x^6+x^3+x^2+1, \quad x^8+x^6+x^4+x^3+x^2+x+1 \\
x^8+x^6+x^5+x+1, \quad x^8+x^6+x^5+x^2+1 \\
x^8+x^6+x^5+x^3+1, \quad x^8+x^6+x^5+x^4+1 \\
x^8+x^6+x^5+x^4+x^2+x+1, \quad x^8+x^6+x^5+x^4+x^3+x+1 \\
x^8+x^7+x^2+x+1, \quad x^8+x^7+x^3+x+1 \\
x^8+x^7+x^3+x^2+1, \quad x^8+x^7+x^4+x^3+x^2+x+1 \\
x^8+x^7+x^5+x+1, \quad x^8+x^7+x^5+x^3+1 \\
x^8+x^7+x^5+x^4+1, \quad x^8+x^7+x^5+x^4+x^3+x^2+1 \\
x^8+x^7+x^6+x+1, \quad x^8+x^7+x^6+x^3+x^2+x+1 \\
x^8+x^7+x^6+x^4+x^2+x+1, \quad x^8+x^7+x^6+x^4+x^3+x^2+1 \\
x^8+x^7+x^6+x^5+x^2+x+1, \quad x^8+x^7+x^6+x^5+x^4+x+1 \\
x^8+x^7+x^6+x^5+x^4+x^2+1, \quad x^8+x^7+x^6+x^5+x^4+x^3+1
\end{gathered}
$$

We have 30 distincts polynomials, but some of them will give equivalent construction.