# Theoretical Background on Cryptographic Primitives

Bogdan Groza

This material intends to be a brief introduction to symmetric and asymmetric cryptographic primitives, pointing out some relevant design principles and security properties. Nonetheless, we call attention to the correct practical use and current standards. This material is intended in part to serve as theoretical background for practical laboratory works were you will get accustomed with the use of cryptographic functions in a more practical way, e.g., by using cryptographic libraries in some programming environment or by analysing protocols.

# 1 Theoretical background on cryptographic primitives

We begin with symmetric cryptographic constructions, i.e., functions that rely on the same key for both encryption and decryption. The fact that the same key is used for both operations should not be interpreted in a strict way, the keys can actually be distinct but it is mandatory that one can be easily computed from the other (in contrast to assymetric primitives). We also group in this sections a class of primitives that do not require a key: hash functions. The reason for this is that they rely more or less on the same design principles and more, they form the basis of the keyed primitive called Message Authentication Code (MAC). A final subject that we encounter is the construction of some pseudo-random number generators, we stay here to a very shallow approach that exemplifies some basic constructions.

Subsequently, we address asymmetric functions, that is, functions that rely on two distinct keys for encryption and decryption (or signing and verification). Since one of the keys (the decryption or signing key) is always kept secret this is also referred as the private key, while its counterpart is made publicly available, hence the name public key. Moreover, computing the private part of the key from the public part must be infeasible.

## 1.1 Symmetric encryption schemes

A symmetric key cryptosystem, also referred to as symmetric encryption scheme, requires the existence of three algorithms: the encryption and decryption algorithms which are an immediate need, but also a key generation algorithm. While the key generation algorithm may do nothing more but picking at random a key, it is core to the security of the scheme and cannot be neglected in the definition (some key generation algorithms do more than picking keys at random as keys need to have a predefined format, this is more prominent in the case of public key primitives). Briefly, the symmetric encryption algorithm takes as input a key $k$ and message $m$ called plaintext and returns the encrypted message $c$ called ciphertext, this is illustrated in Figure 1. Formally, we define a symmetric key cryptosystem as follows.

**Definition 1. Symmetric encryption scheme.** A symmetric encryption scheme is a triple of algorithms:

1. Gen is the key generation algorithm that takes random coins, a security parameter $l$ and outputs the key, i.e., $k \leftarrow \mathsf{Gen}(1^l)$,
2. Enc is the encryption algorithm that takes as input the key and some message, then outputs the ciphertext, i.e., $c \rightarrow \mathsf{Enc}(k, m)$,
3. Dec is the decryption algorithm that takes as input the ciphertext and the key and outputs the message, i.e., $m \rightarrow \mathsf{Dec}(k, m)$.

A correctness condition enforces that always $m \leftarrow \mathsf{Dec}(k, \mathsf{Enc}(k, m))$ In some cases, the encryption and decryption algorithms are allowed to return a special symbol $\perp$ on particular inputs (i.e., they refuse to encrypt/decrypt).

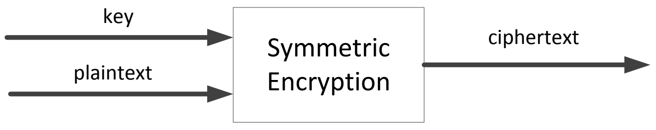


Figure 1: Symmetric encryption algorithm

There are numerous examples of use for symmetric encryption schemes. They are universally present in encrypted tunnels, e.g., SSL/TLS, IPSEC, etc., used to encrypt passwords (e.g., the *lmhash* in Win XP) or to encrypt data on the local hard drive (e.g., TrueCrypt). The current standard is the AES (Advanced Encryption Standard) which covers three key-lengths of 128, 194 and 256 bits. 3DES is a surviving embodiment of the older DES (Data Encryption Standard) which uses keys of 168 bits and is currently considered secure but it is less efficient than the AES, a reason for which it should be avoided in practice. DES is known to be insecure (its key can be easily discovered by exhaustive search) and must be avoided at all costs. RC4 is a stream cipher commonly present in practice due to its elegant and efficient construction but has several security weaknesses for which it should also be avoided.

**Design principle: substitutions and transpositions, product ciphers**. Substitutions and transpositions are the usually employed operation in constructing symmetric cryptosystems. Substitution boxes (S-Box) replace a symbol from the plaintext (or group of symbols) by another symbol (not necessarily from the plaintext) to create confusion. Permutation boxes (P-Box) also known as transpositions mix the symbols inside a block of plaintext to create diffusion. Ciphers that use both substitutions and permutations are also called product ciphers. Some authors use the term product ciphers to denote any cipher that uses more than one transformation, while product ciphers that use only substitutions and permutations are called SP-networks. DES and AES, the two well known standards, are product ciphers, the Feistel cipher which stays behind DES is also product cipher.

**Classification: block ciphers vs. stream ciphers**. A relevant classification of symmetric cryptosystems accounts for the distinction between stream ciphers and block ciphers. In stream ciphers, the message is combined via a simple transformation (e.g., XOR) with the keystream (which is a pseudo-random stream generated by a more complex mechanism). This message-key mixing operation is done one character (bit) at a time. Examples include the RC4 stream cipher used in SSL/TLS or the A5 stream cipher used in GSM. Typically, stream ciphers have low hardware complexity, are fast, but practical instantiations such as RC4 are not always secure. Simplicity of implementation is the reason for which they are used in communication networks, e.g., GSM. In block ciphers, the message is transformed block by block (e.g., 128 bits) via a transformation that is dependent on the key. The standards DES and AES are both block ciphers. Block ciphers can be turned into stream ciphers in certain modes of operations, e.g., counter mode. This means that if you have a secure block cipher you can build a secure stream cipher as well (however this also means that distinction between the two is not always clear and obviously, a stream cipher built this way will not be more efficient than the underlying block cipher).

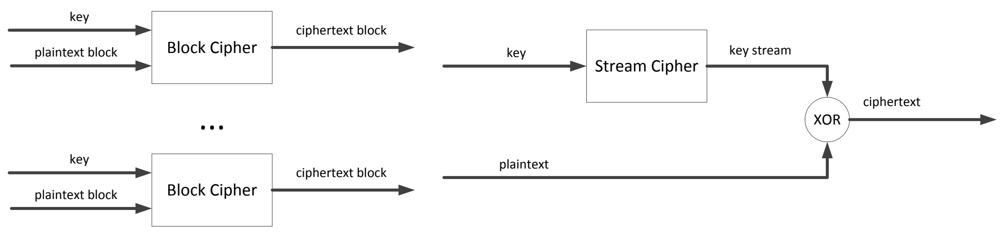


Figure 2: Block cipher (left) vs. stream cipher (right)

Construction 1. The One-time Pad. The one-time pad is a stream cipher in which a random key with the same length as the plaintext is simply XORed with the plaintext. The key must not be re-used and it is randomly generated at each encryption.

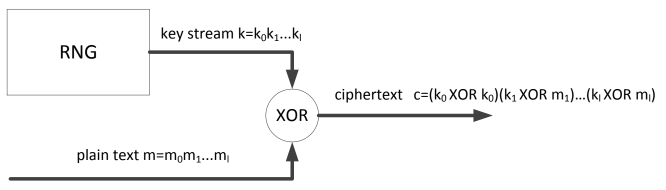


Figure 3: The one-time pad

Believe it or not, the one-time pad is information-theoretically secure, that is, it cannot be broken regardless of the computational power and the quantity of ciphertext available. But obviously there is a big practical problem: it requires a random key of the same length as the plaintext. However, in practice you want a key as small as possible, optimally one that you could even remember and mandatory one that is easy to store in the memory of a device or sent over the network. Since the key needs to be exchanged a-priori on a secure channel (assuming a scenario in which you want to send encrypted messages between two endpoints), if the key has the same length as the message then why not simply exchange the plaintext? This makes the OTP currenlty almost absent in practice. There are still some relevant practical application for it, e.g., quantum cryptography, and its theoretical value must not be neglected.

Construction 2. Feistel network. The Feistel network is a block cipher (and also an S&P network). Each block is split into a right and left part (if equal in size, then the network is called balanced). The right block is passed through a round function $f$ that depends on the round key $k_i$ and XORed with the left block, then the blocks are switched for the next round. The round key is derived from the master key via a key-scheduling algorithm. The number of rounds depends on the designer's choice, for example DES uses 16, while less than 3 rounds cannot assure security. Figure 4 depicts this construction.

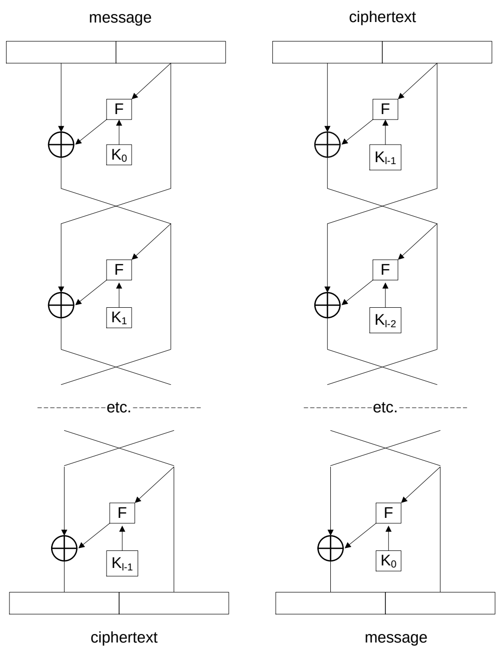


Figure 4: The Feistel network

The Feistel network is the work of Horst Feistel in the 70s at IBM. It stays at the core of DES and its relevance stems from the fact that it provides an efficient security/performance trade-off: by increasing the number of rounds and the size of the key, the security level is increased. That is, you can simply add more rounds and lengthen the key if you feel insecure, anybody can make more secure encryptions this way (but not more efficient). Decryption is performed by walking through the circuit in reverse order, this leads to efficient hardware implementations. Note that the Feistel round is invertible regardless of the properties of the round function, so inverting the network is straight-forward. We prove this in what follows. By definition, deriving the output from the input is done by computing:

$$L_i = R_{i-1}, R_i = L_{i-1} \oplus f_i(R_{i-1}).$$

Which implies, deriving the input from the output can be done by computing:

$$R_{i-1} = L_i, L_{i-1} = R_i \oplus f_i(L_i).$$

Construction 3. Data Encryption Standard (DES). DES is a 16 round balanced Feistel network that operates on 64 bit blocks (split into two 32 bit parts). It uses a 56 bit key from which 16 round keys (48 bit each) are derived. Each round works as follows: the right half (32 bits of the message block) is expanded to 48 bits then XORed with the round key of 48 bits, each 6 bits are then provided as input to 8 S-Boxes that output only 4 bits resulting in 32 bits that are passed through another permutation P. This round transformation is applied 16 times, each time with a distinct round key which is derived from the master key. Figure 5 gives an overview of DES, Figure 6 shows the round function and Figure 7 illustrates the key derivation function.

DES was developed in the 70s at IBM based on Feistel's design and standardized with the input from NSA, then stayed as symmetric encryption standard between from 1977 to 2001. It was considered insecure since the end of the 90s but replaced by the Advanced Encryption Standard (AES) only in 2001.

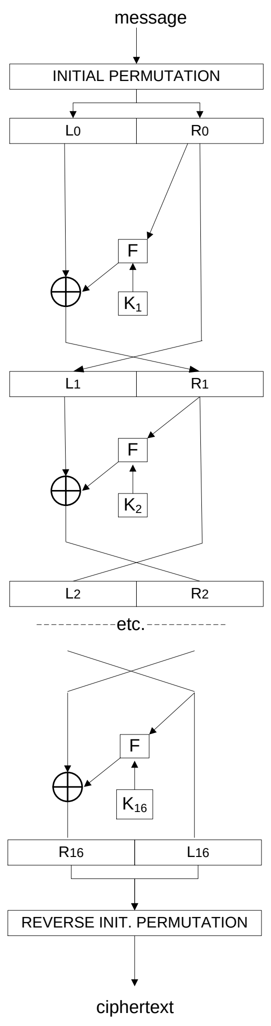


Figure 5: Overview of DES

There are at least two interesting DES oddities. These are not considered weaknesses since if random keys are chosen it is very unlikely (negligible in fact) for these to affect security. First, DES has four weak keys for which encryption and decryption have the same effect, i.e., if you encrypt twice with any of these keys the message stays unencrypted. Second, DES has six pairs of semi-weak keys: encryption with one key from the pair behaves as decryption with the other.

Construction 4. Triple DES (3DES). 3DES consists in the application of DES 3 times with keys $k_1$, $k_2$, $k_3$ as follows: $c = \mathsf{Enc}_{k_3}(\mathsf{Dec}_{k_2}(\mathsf{Enc}_{k_1}(m)))$. Decryption goes the expected way: $m = \mathsf{Dec}_{k_1}(\mathsf{Enc}_{k_2}(\mathsf{Dec}_{k_3}(c)))$. According to the standard, 3DES has 3 keying options: i. independent keys, ii. $k_1$ and $k_2$ independent but $k_3 = k_1$, iii. all keys are equal $k_1 = k_2 = k_3$ (this is DES).

3DES is considered to be secure so far (in the three random keys instantiation) but it is slower than AES and thus there are no serious reasons for using it in practice. One of the reasons for its persistence in practice persistence may be the fact that it is easy to implement in hardware, the electronic payment industry standardized on DES during the 90's and now 3DES is present in all smart-cards.

Construction 5. Advanced Encryption Standard (AES). Operates on a 4x4 matrix of bytes (128 bit blocks) called state. It has 10, 12 or 14 rounds according to the key size 128, 192 and 256 bits. Each round has 4 transformations:

1. SubBytes (a substitution) is non-linear substitution where each byte is replaced via a look-up table,

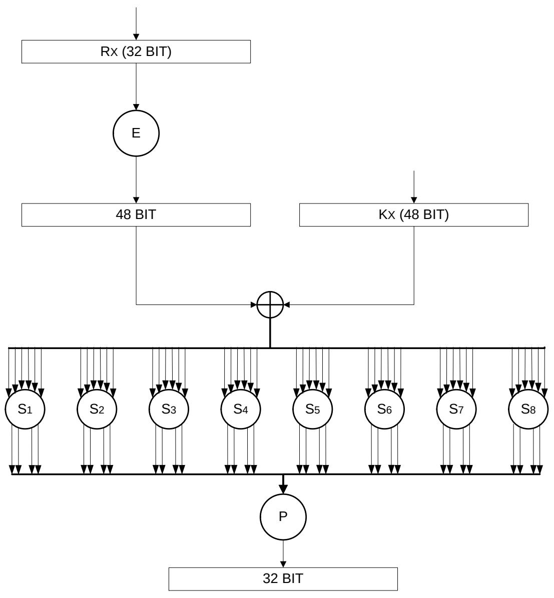


Figure 6: DES round function

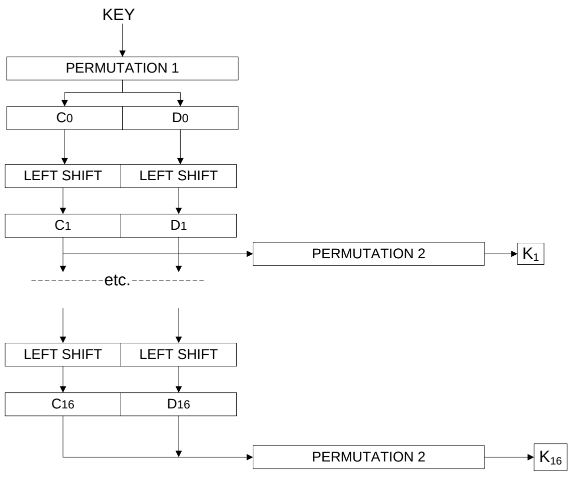


Figure 7: DES key scheduling

2. ShiftRows (a permutation) the last three rows are shifted,
3. MixColumns the four bytes of each column are combined via a linear transformation,
4. AddRoundKey each byte of the state is combined with the round key via a XOR operation.

AES was designed by Vincent Rijmen and Joan Daemen and selected by public competition from the 5 finalists: MARS, RC6, Rijndael (the AES), Serpent, and Twofish. AES is the new standard as of 2001. Note that AES is not a Feistel network.

**Modes of operation**. Block ciphers can encrypt a single message block at a time, to encrypt messages of arbitrary size several modes of operations can be used. These modes of operation affect both the security and efficiency, so chosing a particular mode of operation must be judiciously done.

Construction 6. Electronic Code Book (ECB). In this mode of operation, the message is parsed into blocks and each block is independently encrypted with the secret key, see Figure 8. Decryption is done by reversing this operation, see Figure 9. As expected, last message chunk is padded to the block length.

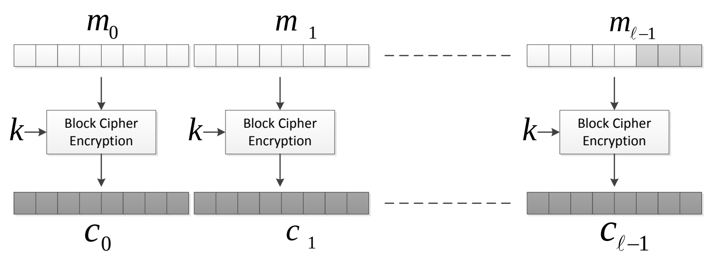


Figure 8: Encryption in Electronic Codebook (ECB)

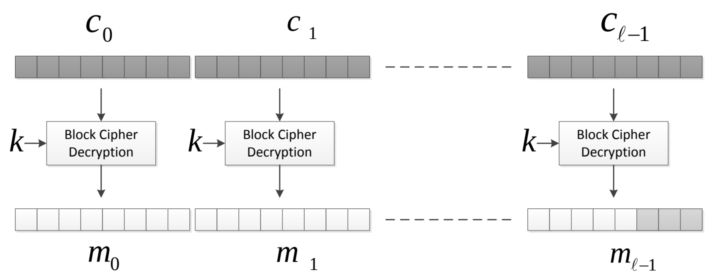


Figure 9: Decryption in Electronic Codebook (ECB)

***Question 1.*** *Assuming that the Block Cipher is secure, is ECB a secure mode of operation?*

*No. This answer will also depend on how you define security, we will dedicate a forthcoming section to this, but even in the more intuitive way of stating that the encryption should not leak any details about the message it is easy to see that ECB fails to do so. The reason is that blocks with similar content will encrypt to the same value, thus it is easy to identify patterns in the encrypted text - while these may not seem much to the reader, it is a serious attack, in practice this may be all that the adversary needs.*

Construction 7. Cipher Block Chaining (CBC). An Initialization Vector (IV) is a non-secret random value used for the randomization of the first output block. In CBC an IV is used to XOR the first message block, before encryption. Subsequent encrypted outputs are XORed with the message before encrypting it, see Figure 10. Decryption is done in reverse order, see Figure 11.

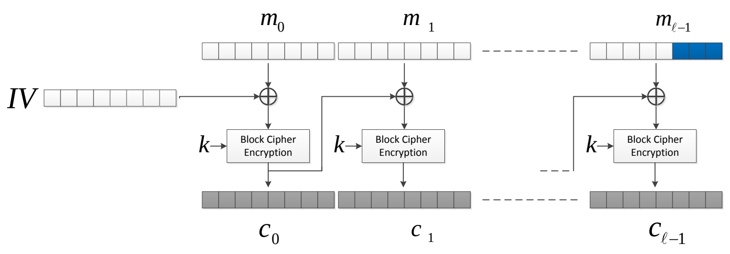


Figure 10: Encryption in Cipher Block Chaining (CBC)

CBC gives a ciphertext that is fully randomized and secure. However, it also comes with a drawback: if one of the blocks is lost, decryption of the next cannot be performed. This is fixed in the following modes of operations. However, it should not be seen as a big drawback for CBC since in practice there are scenarios

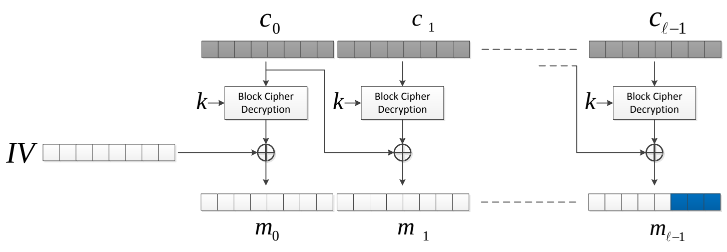


Figure 11: Decryption in Cipher Block Chaining (CBC)

where you can assure a mechanism for reliable transport of the message blocks. In fact CBC is a frequent and correct choice in practice.

Construction 8. Output-feedback (OFB) and Cipher Feedback (CFB). In OFB and CFB the IV is encrypted (instead of the message) then the output is XOR-ed with the message to obtain the ciphertext. The difference between OFB and CFB is in the way of providing the input for the encryption of the next block. OFB uses the input before XOR-ing it with the message, CFB uses the full output. Refer to Figure 12 and Figure 13 for encryption in OFB and CFB modes.

OFB and CFB allow decryption even when message blocks are lost. OFB has an advantage over CFB as it also allows pre-computation of the key stream (this is an important characteristic also present in Counter Mode).

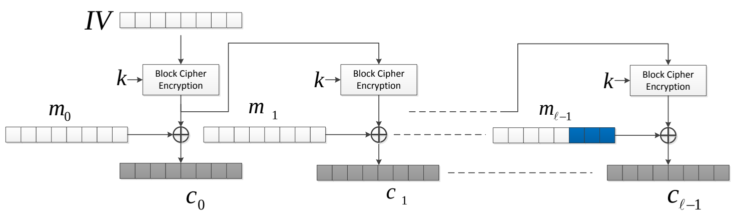


Figure 12: Encryption in Output Feedback (OFB)

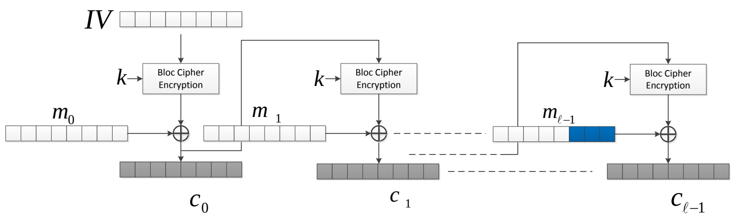


Figure 13: Encryption in Cipher Feedback (CFB)

Construction 9. Propagating Cipher Block Chaining (PCBC). PCBC is a mode of operation similar to CBC that propagates both the inputs and the outputs for encrypting the subsequent block. See Figure 14 and Figure 15 for encryption and decryption in PCBC.

The next mode of operation, Counter Mode, offers unexpected simplicity while preserving security. In CM decryption can still be performed if bocks are lost and more, the key-stream can be pre-computed (this allows buffering which is important for many applications). Basically, CM allows the conversion of a block cipher into a stream cipher.

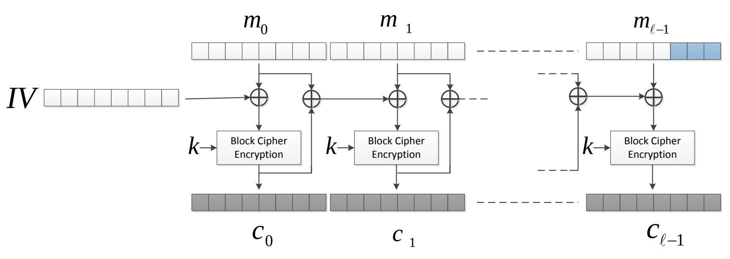


Figure 14: Encryption in Propagating Cipher Block Chaining (PCBC)

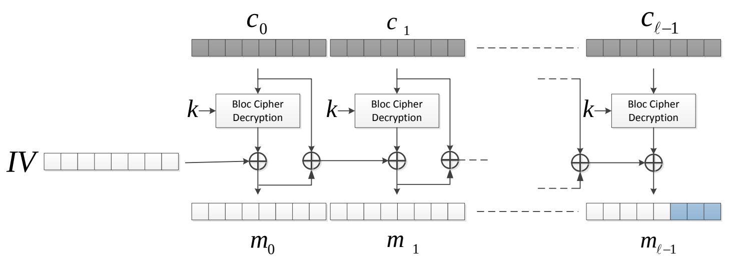


Figure 15: Decryption in Propagating Cipher Block Chaining (PCBC)

Construction 10. Counter Mode (CM). A counter is incremented and encrypted then XORed with each block of the message. See Figure 16 for encryption, a salting value is used to avoid repeating the counter on re-initialization, decryption is straight forward.

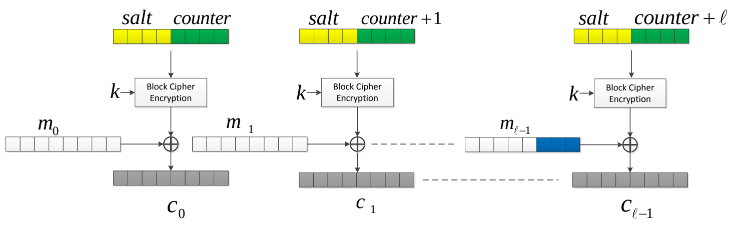


Figure 16: Encryption in Counter Mode (CM)

***Summary.*** *There are several ideas you should remember when working with symmetric ciphers:*

- *do not use DES and RC4 as these are insecure*
- *use AES 128 bit or above, 3DES is secure but it is less efficient so it must be avoided*
- *do not use ECB mode of operation, use CBC or CM*
- *in CBC always pick random IVs, in CM never repeat counters*
- *the Feistel network provides a generic schema for building secure block ciphers*
- *in practice, being secure is not the top goal, being* ***efficient and secure*** *is what matters most*

## 1.2 Adversary models, security properties and reductions

**Adversary capabilities (CPA and CCA).** Before formally defining the security requirements for symmetric encryptions we need to clarify adversary's capabilities, i.e., what the adversary can do. Current security definitions acknowledge the existence of two kinds of adversaries:

- Chosen-plaintext Adversary (CPA), an adversary that has access to a black-box that encrypts plaintexts at the adversary's choice
- Chosen-ciphertext adversary (CCA), an adversary that has access to a black-box that decrypts cyphertexts at the adversary's choice.

**Adaptive vs. non-adaptive adversaries**. An additional flavour that can be added to both CPA and CCA is whether the adversary can continue (adaptive) or not (non-adaptive) to query the encryption and decryption boxes after he received the target ciphertext that he is required to break. Obviously, the adversary is not allowed to query the target ciphertext to the decryption box since in this case the attack would be trivial.

**Security notions**. There are at least three security notions worth mentioning for symmetric encryption schemes, informally:

- Semantic security (SS), introduce by Goldwasser and Micali in 1982 is a fundamental requirement that any information that can be efficiently computed with the ciphertext, can be also computed without the ciphertext,
- Indistinguishability of ciphertexts (IND), requires that given two messages selected by the adversary and the encryption of one of them chosen at random (without adversary's knowledge) the adversary cannot decide which is the encrypted message,
- Real or random indistinguishability (RoR), requires that given a plaintext selected by the adversary and the encryption of either this message or some complete random message (not known to the adversary) the adversary cannot decide if the ciphertext corresponds or not to its chosen plaintext (we assume that the random and the real message have the same length).

Under proper formalization all these notions are equivalent, see Goldreich [6], p.383. Generally IND or RoR are easier to prove and are the standard tool in proving security.

**Security reductions**. Reductions, showing that solving a problem reduces to solving another one, are the tool for proving that one security property implies another. The answer to the next question requires you to build a reduction from RoR to IND and vice-versa.

***Question 2.*** *Given a CPA adversary, prove that a symmetric encryption scheme is resilient to RoR if and only if it is resilient to IND.*

***Summary.*** *Indistinguishability of encryptions (or equivalently real-or-random indistinguishability) under adaptive chose-ciphertext adversaries is the desired security goal when designing symmetric-key cryptosystems.*

## 1.3 Hash functions

Hash functions are algorithms that take as input a message of any length and turn it into a constant size output (usually referred to as tag or hash of the message). They have a large variety of use cases: generally to assure integrity of software downloads/updates, to protect stored passwords, etc. The current standard for the Secure Hash Algorithm (SHA) is still the SHA2 which has 3 output variants on 256, 384 and 512 bits. Usually the 256 bit output version should be employed, the 384 one is as slow as the 512 bit one, currently there are no known weaknesses of SHA2. The use of MD5 and SHA1 should be avoided as these are no longer resistant to collisions (that is, messages that have the same output are known). For future use the SHA3 is under development (Keccak is the winner of the 2013 competition).

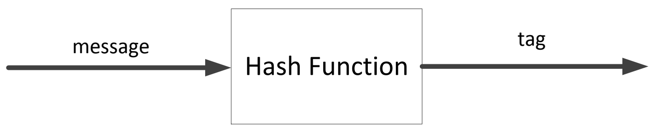


Figure 17: Hash Function

**Security properties for hash functions**. The following security properties are mandatory for hash functions:

- Pre-image resistance, i.e., given the hash of some message it is infeasible to find the message, i.e., $H(m) \rightarrow m$
- Secondary pre-image resistance, i.e., given the hash of a message and the message it is infeasible to find a second message that has the same hash value, i.e., $\{m_0, H(m_0)\} \rightarrow \{m_1, H(m_0) = H(m_1)\}$
- Collision resistance, i.e., it is infeasible to find two messages that have the same hash, i.e., $\rightarrow \{m_0, m_1, H(m_0) = H(m_1)\}$

The Merkle-Damgard construction provides a method for turning a collision-resistant one-way function into a collision-resistant hash functions. This design stands behind MD5, SHA1 and SHA2.

Construction 11. Merkle-Damgard Construction. An IV is fixed which does not need to be random like in block ciphers mode of operation. This IV along with the first message block is passed through a compression function and the output is provided as IV for the second block, etc. See Figure 18 for the Merkle-Damgard construction.

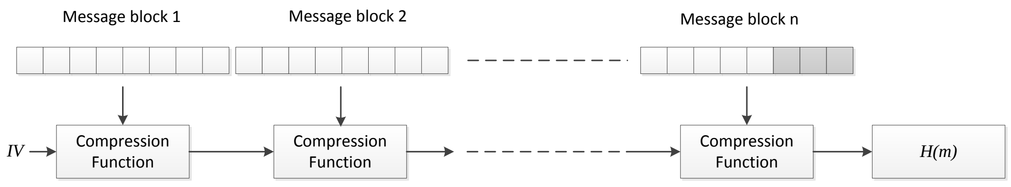


Figure 18: The Merkle-Damgard construction

Construction 12. Message Digest 5 (MD5). MD5 is a Merkle-Damgard based construction. It uses 4 IVs defined as follows: A = 0x67452301, B = 0xefcdab89, C = 0x98badcfe, D = 0x10325476. The message is processed in blocks of 512 bits that are further split in 128 bit chunks and propagated as IVs for the next block to be hashed. There are 4 rounds, each of them applies the round function 16 times to 16 input bytes (i.e., the Merkle-Damgard block is 64 bytes). Each of the four rounds does the following transformation 16 times: $D \leftarrow C$, $C \leftarrow B$, $B \leftarrow B \oplus ((A \oplus F(B, C, D) \oplus M \oplus K) <<< S)$, $A \leftarrow D$ (A, B, C, and D are the IVs, K and S are fixed constants and M is the message byte). The round function is distinct for each of the four rounds, still, all round functions consist in simple logic operations $\wedge$, $\vee$, $\oplus$ and $\neg$ (which stay for AND, OR, XOR and NOT). The four round functions are defined as: $F(B, C, D) = (B \wedge C) \vee (\neg B \wedge D)$, $G(B, C, D) = (B \wedge D) \vee (C \wedge \neg D)$, $H(B, C, D) = B \oplus C \oplus D$, $I(B, C, D) = C \oplus (B \vee \neg D)$. Figure 19 gives an overview of MD5.

To give you an example of what you get after you hash, Table 1 shows the test vectors as defined by RFC 1321.

| | |
|---|---|
| MD5 ("") | = d41d8cd98f00b204e9800998ecf8427e |
| MD5 ("a") | = 0cc175b9c0f1b6a831c399e269772661 |
| MD5 ("abc") | = 900150983cd24fb0d6963f7d28e17f72 |
| MD5 ("message digest") | = f96b697d7cb7938d525a2f31aaf161d0 |
| MD5 ("abcdefghijklmnopqrstuvwxyz") | =c3fcd3d76192e4007dfb496cca67e13b |
| MD5 ("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789") | = d174ab98d277d9f5a5611c2c9f419d9f |
| MD5("12345678901234567890123456789012 3...234567890") | = 57edf4a22be3c955ac49da2e2107b67a |

Table 1: MD5 test vectors as defined in RFC 1321

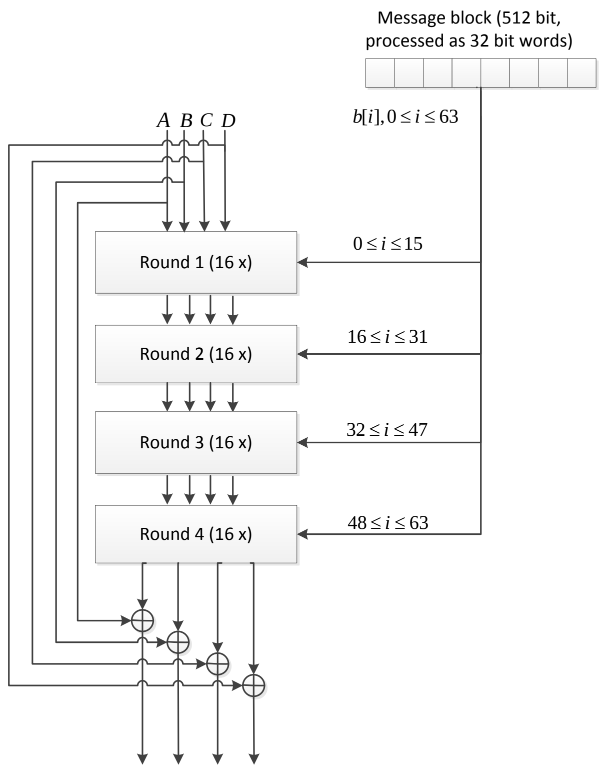


Figure 19: The MD5 Hash Function

***Summary.***

- *MD5 and SHA1 should be avoided in practice as they are not collision free*
- *SHA2 is the current (still secure) standard, it will be replaced by SHA3*
- *Merkle-Damgard provides a generic schema for building hash functions*

## 1.4 Keyed hash functions (Message Authentication Codes)

Message Authentication Codes (MAC) are algorithms that take a message of arbitrary length and a key then output a tag. Since in practice they are commonly build on a hash function with a random key, they are usually referred as keyed hash functions. The general use is for assuring message authentication, i.e., binding a message with the identity of a principal that knows the key. Current standards are the HMAC and the NMAC which can be applied with any of the standard hash functions. The simple hash over the concatenation of a key with the message should be avoided as this is in general insecure.

key
plaintext
Message Authentication Code
tag

Figure 20: Message Authentication Code (MAC) algorithm

**Definition 2. Message authentication Code.** A Message Authentication Code is a triple of algorithms:

- Gen is the key generation algorithm that takes random coins, a security parameter $l$ and outputs the key, i.e., , $k \leftarrow \mathsf{Gen}(1^l)$,

- Mac is the tag-generation algorithm that takes as input the key and some message, then outputs the tag i.e., $\tau \leftarrow \mathsf{MAC}(k, m)$,
- Ver is the verification algorithm that takes as input the key, the tag and the message and outputs 1 if the tag is valid or 0 otherwise, i.e., $\{0, 1\} \leftarrow \mathsf{Ver}(k, \tau, m)$.

A correctness condition enforces that it always holds $\mathsf{Ver}(k, \mathsf{MAC}(k, m), m) = 1$.

**Desired security property for MACs**. Fortunately, there is only one strong definition of security that MACs must achieve, they must have (existential) unforgeability under chosen message attacks (of course, this can be refined in several ways). That is, an adversary that receives any number of valid message-tag pairs (i.e., pairs that are computed with the MAC algorithm) is unable to output a new message-tag pair that will successfully pass through the verification algorithm

***Question 3.*** *Based on the previous security definition for MAC codes, is a hash over the simple concatenation of the message with the key, i.e., $H(k||m)$, secure?*

*No. Concatenation attacks are possible due to the construction of the hash function, e.g., in the Merkle-Damgard construction.*

Construction 13. HMAC. The HMAC consists in the application of a hash function twice: first with an inner-padding (ipad) then with an outer-padding (opad), concretely: $\mathrm{HMAC}(K, m) = H((K \oplus opad)||H((K \oplus ipad)||m))$. The ipad is B blocks of 0x36 and the opad is B blocks of 0x5C, where B is the byte size of the block to be processed (e.g., B=64 in case of MD5 that uses blocks of 512bits).

The HMAC can be implemented over any hash function, e.g., HMAC-MD5, HMAC-SHA256, etc. NMAC (Nested MAC) is as simple as HMAC, however it requires changing the IVs which is less handy when implementing, thus NMAC is more absent in practice.

**Authenticated encryption**. There are various paradigms of combining MACs with encryptions to obtain authenticated encryption, i.e., assuring that an encrypted ciphertext indeed originates from the source (note that block ciphers are not designed for assuring authentication and they never do so).

Construction 14. Authenticated Encryption. There are at least three paradigms employed in practice for authenticated encryption:

- Encrypt-and-MAC, encrypt the message and concatenate it with the MAC of the message, i.e., $E_k(m)||MAC_k(m)$,
- MAC-then-encrypt , encrypt the message concatenated to its MAC, i.e., $E_k(m\ ||MAC_k(m))$,
- Encrypt-then-MAC, encrypt the message then compute the MAC of its encryption and concatenate the two, i.e., $E_k(m)||MAC_k(E_k(m))$.

Encrypt-then-MAC has better security properties than the previous two and should be the desired alternative in practice. Encrypt-and-MAC is used in SSH, MAC-then-encrypt is used in SSL/TLS and Encrypt-then-MAC is used in IPSec. For details, please see Bellare and Namprempre in [1].

***Summary.*** *Ideeas you should remember for MAC codes:*

- *do not assume that encryption or hashing will assure authenticity, they are not designed for this, but MACs are,*
- *simple hashing of the concatenation of key plus the message is insecure in general,*
- *HMAC is a secure and efficient MAC construction.*

## 1.5 Random number generators

Random numbers stay at the core of any cryptosystem since you need randomness for the secret keys. True Random Number Generators (TRNG) output random sequences based on physical processes that are hard/infeasible to model, i.e., white noise from a Zenner diode, oscillator drift, SRAM state at power-up, etc. Pseudo-random number generators (PRNGs) are deterministic algorithms that generate a random sequence based on a value called seed. While all PRNGs have cycles, this does not mean that they are always insecure, computationally secure PRNGs exist. Examples of use are numerous, in particular they are used in any handshake such as SSL/TLS, IPSec, etc., that needs to generate a fresh session key.

The linear congruential generator and Linear Feedback Shift Registers are two insecure and yet commonly used (hopefully not for cryptographic purposes) constructions.

Construction 15. The linear congruential generator. Set $x_0$ as a random seed. Then compute the sequence $x_{i+1} = ax_i + c \bmod n$ (where $a, c, n$ are fixed integers).

Construction 16. Galois and Fibonacci Linear Feedback Shift Registers (LFSR). Linear feedback shift registers have a state (the initial state is the seed) and output a bit by shifting the register and updating the next state as a function of the previous state. Galois and Fibonacci registers differ only in the way they XOR the bits for the next state, for Galois the XOR can be done in parallel while for Fibonacci is sequential. Figure 21 shows two 6-bit Galois and Fibonacci registers.

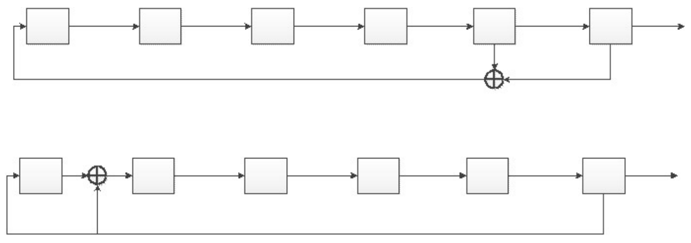


Figure 21: Galois and Fibonacci LFSR

Bloom-Bloom-Shub (BBS) is a cryptographically secure PRNG but it requires a large modulus $n$ and is computationally expensive thus, despite its security, it is almost absent in practice.

Construction 17. Bloom-Bloom-Shub (BBS). Set $x_0$ as a random seed. Then for a large modulus $n$ infeasible to factor compute $x_i = x_{i-1}^2 mod n$ and output the parity bit of this value.

Construction 18. Block ciphers. Use any secure block cipher with a secret random key to encrypt a counter and output the ciphertext, this is a PRNG if the block cipher is secure.

**How to test RNG & PRNGs**. Various statistical tests are usually employed, none is perfect but may provide some degree of confidence. Diehardeer is a battery of tests used by many enthusiasts or professionals http://www.phy.duke.edu/ rgb/General/rand_rate.php

***Summary.*** *Secure PRNGs exists, the problem is having a really random seed and enough computational power (see BBS). For TRNG there are several physical processes that generate unpredictable noise, e.g, Zenner diodes, oscillators, etc.*

## 1.6 Public-key cryptography: definitions and adversary models

A public key encryption algorithm takes as input the public key and the message (called plaintext) then returns the encrypted message (called ciphertext). This algorithm is suggested in Figure 22. A decryption algorithm is also part of the scheme, it takes as input the private key and the ciphertext and returns the message. Similarly to the case of symmetric encryption schemes, a key generation algorithm is needed, this time the procedure is usually more involving than simply selecting bits at random. The formal definition of the public-key cryptosystem follows.

**Definition 3. Public-key (asymmetric) encryption scheme.** An asymmetric encryption scheme is a triple of algorithms:

1. Gen is the key generation algorithm that takes random coins, the security parameter $l$ and outputs the public and private key, i.e., $\{\mathrm{Pb}, \mathrm{Pv}\} \leftarrow \mathsf{Gen}(1^l)$,

2. Enc is the encryption algorithm that takes as input the public key and the message, then outputs the ciphertext, i.e., $c \rightarrow \mathsf{Enc}(\mathrm{Pb}, m)$,

3. Dec is the decryption algorithm that takes as input the ciphertext and the private key and outputs the message, i.e., $m \rightarrow \mathsf{Dec}(\mathrm{Pv}, m)$.

A correctness condition enforces that $m \leftarrow \mathsf{Dec}(\mathrm{Pb}, \mathsf{Enc}(\mathrm{Pv}, m))$.

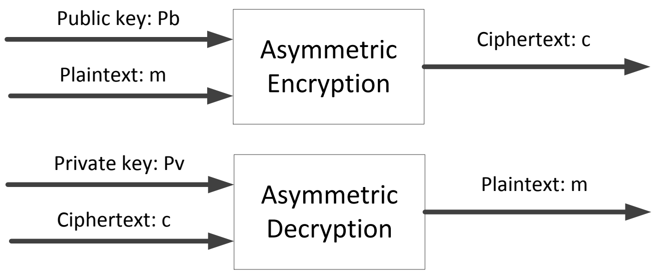


Figure 22: Asymmetric encryption and decryption

There are numerous examples of use for public-key cryptosystems. The main example are likely the key-exchange protocols for encrypted tunnels SSL/TLS, IPSEC, etc. Standards include RSA (recommended at least 2048 bit), Diffie-Hellman (with or without ECC), etc. Small key versions or unpadded (textbook) versions of all public-key algorithms must be avoided. In the near future, it is likely that ECC will completely replace RSA, for years the dissapearance of RSA from practice was predicted but while ECC has continuously grown, RSA is still there.

**Security properties for PKC**. A basic security condition enforces that given the public key Pb it is infeasible to compute the private key Pv, but this is not enough. Similar to what we defined in case of symmetric encryptions we are in a scenario with CPA and CCA adversaries and the two main properties defined for these schemes are:

- Indistinguishability of ciphertexts (IND), identical to the case of symmetric cryptosystems, the adversary must be unable to distinguish between the encryptions of two distinct messages,

- Non-malleability of ciphertexts (NM), here the adversary cannot modify a given challenge ciphertext such that it decrypts to a distinct but valid plaintext.

In Figure 23 we shows relations among security notions for PKC as proved by Bellare, Desai, Pointcheval & Rogaway in 1998. The idea that should be remebedered is that IND-CCA2 is equivalent to NM-CCA2 (CCA2 distinguishes the adaptive adversary from the non-adaptive one in CCA1).

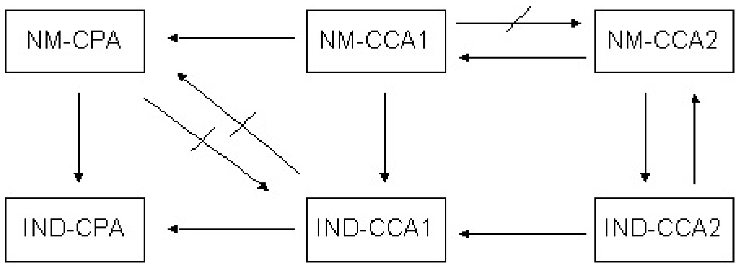


Figure 23: Relations among security notions for PKC

Digital signatures are the electronic equivalent of a handwritten signature, the signing algorithm takes the private key and the message and returns the signature, the verification algorithm takes the public key, the message and the signature and checks if the input is genuine. This is suggested in Figure 24. A key generation algorithm is also needed. General examples include document signing, driver signing, certificate signing, etc. There are several standardized algorithms for signatures RSA-PSS, RSA-PKCS, DSA, etc. Again, small key versions of these algorithms or unpadded (textbook) versions must not be used.

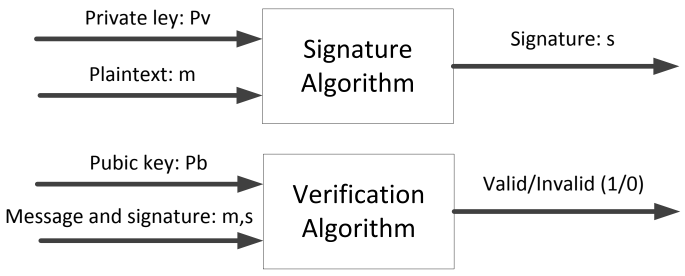


Figure 24: Digital signature and verification

**Definition 4. Digital signature scheme.** A digital signature scheme is a triple of algorithms:

1. Gen is the key generation algorithm that takes random coins, the security of level $l$ and outputs the public and private key, i.e., $\{\text{Pb}, \text{Pv}\} \leftarrow \mathsf{Gen}(1^l)$,
2. Sig is the signing algorithm that takes as input the private key and some message, then outputs the signature, i.e., $s \rightarrow \text{Sig}(\text{Pv}, m)$,
3. Ver is the verification algorithm that takes as input the signature and the public key and outputs the message, i.e., $\{0, 1\} \rightarrow \text{Ver}(\text{Pb}, m)$ and outputs 1 if the signature is valid or 0 otherwise.

A correctness condition enforces that $1 \leftarrow \text{Ver}(\text{Pb}, \text{Sig}(\text{Pv}, m))$.

**Security notions for digital signatures**. Again, a security condition enforces that given the public key Pb it is infeasible to compute the private key Pv, but this is not enough. For digital signatures, usually four distinct levels of forgery are considered:

- Existential forgery, in which the adversary is required to find a valid message-signature pair without controlling the message,
- Selective forgery, in which the adversary is required to forge signatures over messages that have a particular structure,
- Universal forgery, in which the adversary is required to forge signatures over any kind of messages (without knowing the private key),
- Total break, in which the adversary is required to recover the private key (i.e., is able to sign anything).

Adversary capabilities against digital signatures can also be refined along several lines:

- Key-only, in which the adversary knows only the public key,
- Known-messages, in which the adversary has valid messages-signature pairs but not at his choice,
- Chosen message, in which the adversary has message-signature pairs at his choice (adaptive chosen-message is a flavour of this notion where the adversary is allowed to chose messages after fixing the target to be forged).

To sum up, unforgeability under chosen-message attacks is the desired property, this means that the adversary cannot forge signatures, even if he has full access tot the signing oracle.

***Summary.***

- *IND and NM under CCA adversaries are the desired security properties for asymmetric encryption schemes*
- *unforgeability under CCA adversaries is the desired property for signature schemes*

## 1.7 The RSA encryption

The following construction presents the RSA cryptosystem, the work of Rivest, Shamir and Adleman which is likely the best known cryptosystem today [7] (best known, but still surrounded by open problems such as its un-proved equivalence to factoring).

Construction 19. RSA encryption (textbook version). The RSA public-key cryptosystem is the triple of the following algorithms:

1. $\mathsf{Gen}(1^l)$ is the key generation algorithms. It selects two random primes $p, q$ (each of $l/2$ bits), computes $n = pq, \phi(n) = (p-1)(q-1)$, chooses $e$ relatively prime to $\phi(n)$, compute $d$ such that $ed \equiv 1 \mathrm{mod} \phi(n)$. The public key is $\mathrm{Pb} = (n, e)$ and the private key is $\mathrm{Pv} = (n, d)$,
2. $\mathsf{Enc}(m, e, n)$ is the encryption algorithm, it computes $c = m^e \mathrm{mod} n$ (note that the message must be represented as integer mod n),
3. $\mathsf{Dec}(c, d, n)$ is the decryption algorithm, it computes $m = c^d \mathrm{mod} n$.

**Real World RSA Keys**.. Since large keys are used for RSA in the real-world, it may be intuitive to take a look in Figure 25 at a 2048 bit RSA key. This key is from the RSA Factoring Challenge, the prize offered for its factorization was 200.000$, while it currently remains unfactored, the competition is now closed.

251959084756578934940271832400483985714292821262040320277771378360436620207075955562640185258807
844069182906412495150821892985591491761845028084891200728449926873928072877767359714183472702618
963750149718246911650776133798590957000973304597488084284017974291006424586918171951187461215151
726546322822168699875491824224336372590851418654620435767984233871847744479207399342365848238242
811981638150106748104516603773060562016196762561338441436038339044149526344321901146575444541784
240209246165157233507787077498171257724679629263863563732899121548314381678998850404453640235273
81951378636564391212010397122822120720357

Figure 25: Example of 2048 RSA key from RSA factoring competition (closed)

***Question 4.*** *Consider to factor by exhaustive search? What is the expected number of steps?*

*Since exhaustive search has a worst time complexity of $O(n^{1/2})$ and the above modulus is 2048 bits, one would expect the number of steps to be in the order of $2^{1024}$. You should take a look at the following:*

- *electrons in the known universe: 8,37*1077837000000000000000000000000000000000000000000000 00000000000000000000000000000,*
- *age of solar system: 1,89*1017189000000000000000.*

*Exhaustive search is not an option now. The following rewards were withdrawn by RSA: RSA-768 - \$50,000 USD (factored December 12, 2009), RSA-896 \$75,000 USD, RSA-1024 \$100,000 USD, RSA-1536 \$150,000 USD, RSA-2048 \$200,000 USD. None of these sums can cover exhaustive search on your or any other standard PC.*

**Notes on RSA computational requirements**. Given that RSA operates with large integers, some notes on computational requirements are needed. Generating keys is the most intensive computational step. It requires the generation of two random primes, each needing the generation of a random integer plus testing for primality (since there are around $x/ln(x)$ prime numbers up to $x$, the probability of success is around $1/ln(x)$). Encryption is usually the most efficient step since one can choose special form exponents: 3, 5, 65537 (note that these are primes of the form 10000001). Decryption is always more computationally intensive than encryption because the decryption exponent is in the order of the modulus $n$.

***Question 5.*** *Why are exponents of the form 100001 preferred? Why is the decryption exponent in the order of n?*

**RSA Chinese Remaindering Theorem (CRT) speed-up**. For faster computations, RSA decryption (or equivalently RSA signature) is usually performed with Chinese-Remaindering-Theorem. This allows decryption to be done modulo $p$ and $q$ then combine the results to get the value modulo $n$. This means computing $m_0 = c^{d'} \text{mod} p, m_1 = c^{d''} \text{mod} q$ then $m = (m_0(q^{-1} \text{mod} p)q + m_1(p^{-1} \text{mod} q)p) \text{mod} n$, where $d' = d \text{mod}(p-1), d'' = d \text{mod}(q-1)$. There are alternative ways for doing the same, e.g., see for example the .NET implementation.

***Question 6.*** *Why is the decryption exponent reduced modulo p-1 and q-1? Why this works faster than standard decryption and what is the expected speed-up?*

**Some mathematical properties**. There is no proof of equivalence between breaking RSA and factoring so far. Factoring obviously leads to breaking the RSA, computing a private-public RSA key pair also leads to factoring. Proving that RSA decryption leads to factoring seems to be hard so far, or maybe this equivalence is not true after all.

***Question 7.*** *Consider IND (indistinguishability) as security property, is textbook RSA secure under this property?*

*No, in fact no deterministic public key cryptosystem is. To prove this, simply encrypt one of the challenge plaintexts and compare the result with the target (note that the public key is known, so it is always possible to do this).*

***Question 8.*** *Consider an adaptive CCA adversary, can the adversary recover the full plaintext in case of textbook RSA?*

*Yes, textbook RSA is completely insecure under CCA adversaries. The target of this attack would be for the adversary to recover the plaintext $m_1$ that corresponds to a target $c_1 = m_1^e \text{mod} n$. The problem is that the decryption machine refuses to decrypt this target, however, the adversary can simply multiply $c_1$ with another ciphertext $c_2$ of some known message $m_2$ then feed $c_1 c_2$ as input to the decryption machine. When receiving the output (which is indeed $m_1 m_2$) the adversary simply divides it by $m_2$ to get $m_1$. Figure 26 depicts this attack.*

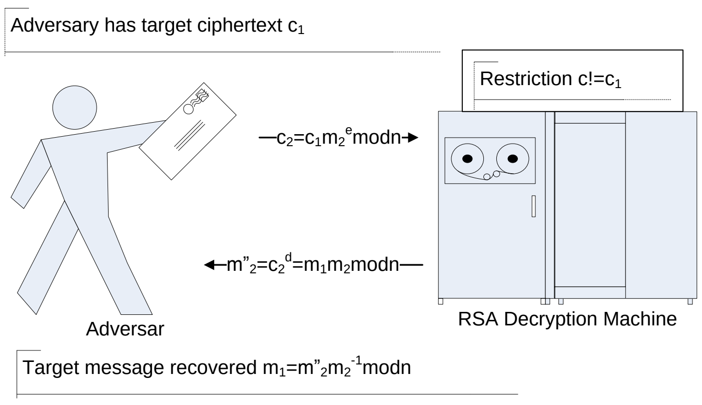

Adversary has target ciphertext $c_1$
Restriction $c!=c_1$
$c_2=c_1 m_2^e modn$
$m''_2=c_2^d=m_1 m_2 modn$
Adversar
RSA Decryption Machine
Target message recovered $m_1=m''_2 m_2^{-1} modn$

Figure 26: Adaptive chosen ciphertext attack on RSA

The Optimal Asymmetric Encryption Padding (OAEP) was proposed by Bellare and Rogaway in 1994 to make RSA resilient against adaptive NM/IND adversaries [2]. There were several historical turnarounds for OAEP: first Bellare and Rogaway proved that OAEP gives such security on any trapdoor, but later Shoup proved they were wrong [8] (obviously this cannot hold for any trapdoor). Soon after, Fujisaki and Okamoto [5] proved that security of OAEP holds for the RSA trapdoor, and thus RSA-OAEP is correct and secure practical construction. A distinct line of criticism is that all proofs for OAEP are in the Random Oracle Model (ROM) which assumes hash functions to behave as pure random functions but in practice hash functions are not random functions. Still the ROM is considered a good model for proofs, and there are no known weaknesses yet in practice for schemes proven in this model.

Construction 20. Optimal Asymmetric Encryption Padding (RSA-OAEP). While not necessarily obvious, the main idea in OAEP is to embed a Feistel network under RSA. For this, regular RSA encryption is done over $m \oplus G(r)||r \oplus H(x \oplus G(r))$. Here $H, G$ stand for some random functions, in practice these are instantiated with hash functions.

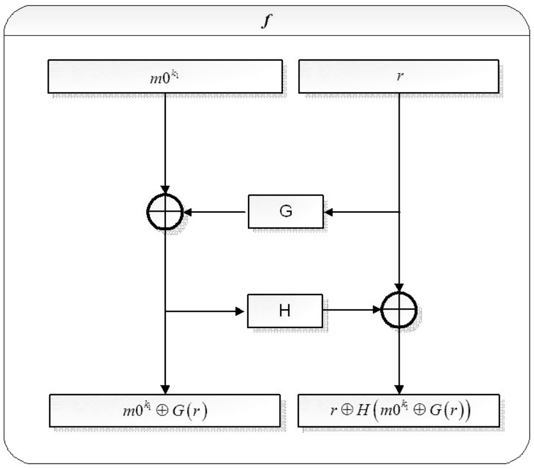


Figure 27: The OAEP padding for RSA

Another padding for RSA to prevent CCA attacks is described in PKCS#1 (Public-Key Cryptography Standards). With this padding the previous CCA attacks does not work, however, there are some attacks for some special cases (small exponents, special messages, etc.), and more, there is no proof that RSA-PKCS#1 is secure. The newer versions of PKCS#1 include RSA-OAEP as improved encryption/decryption method.

Construction 21. RSA-PKCS#1. Before encryption, the message is padded as: 00...00||00...10||||00...00||$m$, that is a byte of value 0 followed by one of value 2 then by a random number of $k - 3 - |m|$ bytes (at least 8) where k is the byte length of the modulus.

**The Rabin cryptosystem**. For brevity we will not insist here on the details, but the Rabin cryptosystem may likely worth a section for itself. Published in 1979 by M.O. Rabin, it has the same description as the RSA but instead of using a public exponent that is co-prime to the order of the group it uses a fixed exponent $e = 2$. This makes it not to be a particular case of RSA since 2 can never be co-prime to $\phi(n)$ (a function that is always even). It comes with a small practical drawback, if the modulus is the product of two primes then there are 4 square roots so then after decryption you need redundancy/padding to decide which of them was the message. This is not a major drawback, but requires more work and may be one of the reasons for which it failed to be adopted in practice. A second reason may be the fact that if you fail to determine de correct message and output one of the other messages, then an adversary that knows the message can with probability 1/2 obtain the factorization of $n$.

***Question 9.*** *Why are there 4 roots of a square modulo n? More, why having two roots of some square modulo n would allows factorization of n?*

***Summary.***

- *to assure security large RSA keys are needed (2048, 4096 bits)*
- *text-book RSA is insecure without padding, thus RSA must not be used without padding*
- *OAEP is a provable secure padding scheme for RSA*

## 1.8 The RSA digital signature

The next scheme provides the basic RSA signature signature. We call this the textbook version of the RSA signature for the same reasons as for the textbook version of the RSA encryption: it is good for learning but insecure in practice. By adding correct padding schemes to it, it can be turned into a secure scheme as we explore next. The improved RSA signature schemes that follow are in fact only padding schemes (the same holds for the real-world versions of the RSA encryption scheme).

Construction 22. RSA signature (textbook version, hash then sign paradigm). The RSA signature scheme is the triple of the following algorithms:

1. $\mathsf{Gen}(1^l)$ is the key generation algorithms. It selects two random primes $p, q$ (each of $l/2$ bits), computes $n = pq, \phi(n) = (p-1)(q-1)$, chooses $e$ relatively prime to $\phi(n)$, compute $d$ such that $ed \equiv 1 mod \phi(n)$. The public key is $\text{Pb} = (n, e)$ and the private key is $\text{Pv} = (n, d)$.
2. $\text{Sig}(m, d, n)$ is the signing algorithm, it computes $s = h(m)^e mod n$ where $h(m)$ denotes the hash of message $m$.
3. $\text{Ver}(s, m, e, n)$ is the verification algorithm, it computes $v = s^e mod n$ then the hash of $m$ and checks that $v = h(m)$.

***Question 10.*** *Consider that in the case of the RSA signature, rather than signing the hash, one signs directly over the message. Show an existential forgery on this signature scheme.*

**Signing as the inverse of encryption**. Note that in case of RSA, the signing algorithm is the reverse of encryption algorithm, this leaves the impression that in general signing is the reverse of encryption, but turns out not to be the case for many other public key cryptosystems, e.g., ElGamal. In general signing should probably not be seen as the inverse of encryption, for the same reasons as MAC codes are not the inverse of symmetric enryptions.

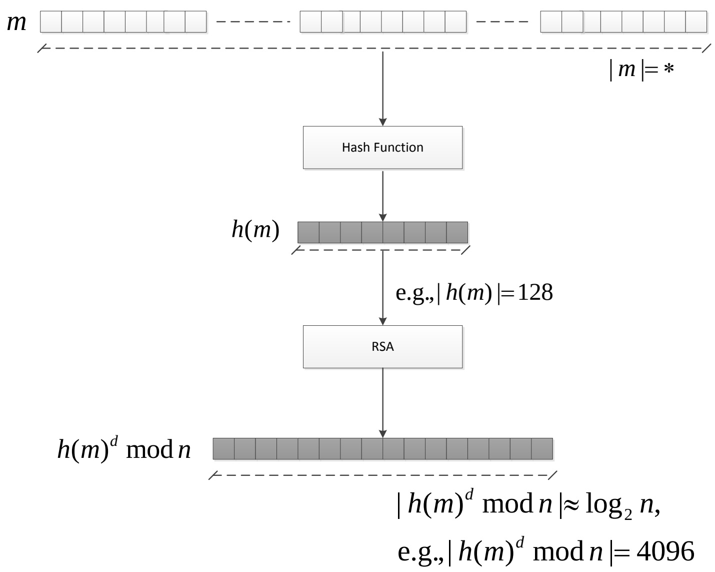


Figure 28: RSA Signature - Textbook Hash then Sign

The RSA full domain hash is a secure version of the RSA which is provable secure in the Random-Oracle-Model. The principle is just to use a hash function that spans over the entire domain of the modulus. While hash functions have smaller outputs, it is really easy to expand the size of the output.

Construction 23. RSA Full Domain Hash (FDH). Identical to text-book RSA except that a hash function with an output of the same size as the modulus is used.

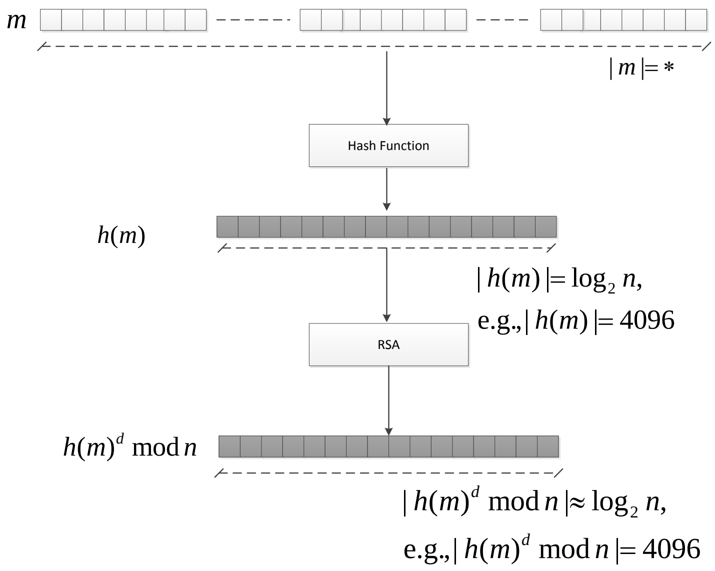


Figure 29: RSA Full Domain Hash

The standard published by RSA laboratories as of 1991 (current version is from 2012) also includes a padding scheme for signing (it is the analogue of the padding scheme for encryption).

Construction 24. RSA PKCS v.1.5 signature scheme. The RSA - PKCS v.1.5 is a padding for the RSA signature as follows. Two bytes of value 0 and 1 followed by a fixed value PS (encoding 0xFFh) followed by a 0 byte are added to the hash written in ASN.1 (Abstract Syntax Notation) which is just an encoding scheme that adds the name of the hashing algorithm to the hash of the message. Note that this is similar to the PKCS encryption padding but the constant is different (for encryption the constant was in fact a random value). This is depicted in Figure 30.

A more involving scheme is the Probabilistic Standard Signature from Bellare and Rogaway [3]. It is a more complex approach but benefits from a tight security proof, an essential element that PKCS v.1.5 is missing.

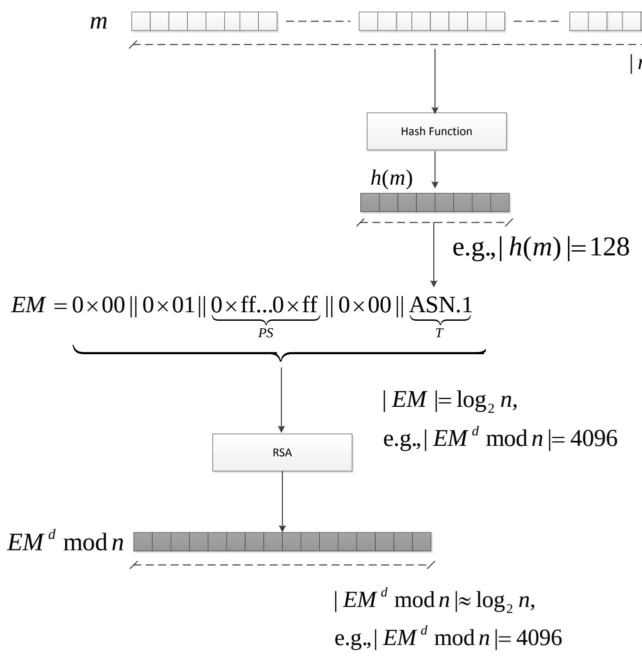
Construction 25. RSA Probabilistic Standard Signature (PSS). PSS uses two paddings and passes the message twice through a hash function, then once through a Mask Generation Function (MGF) which is also a hash function. It also uses a salt value that makes this signature randomized (i.e., multiple signatures for the same message). This is depicted in Figure 31.

***Summary.***

- *similar to RSA encryption, RSA signature without padding is insecure*
- *FDH and PSS are provable secure paddings for RSA signatures*

## 1.9 The Diffie-Hellman Key Exchange

Diffie and Hellman described in 1976 a method for securely exchanging a key over an insecure channel between two parties [?]. This is considered the academic beginning of public-key cryptography. This construction is



$m$
$|m| = *$
Hash Function
$h(m)$
e.g., $|h(m)| = 128$
$EM = 0\times00 \| 0\times01 \| \underbrace{0\times\mathrm{ff}...0\times\mathrm{ff}}_{PS} \| 0\times00 \| \underbrace{\mathrm{ASN.1}}_{T}$
$|EM| = \log_2 n$,
e.g., $|EM^d \bmod n| = 4096$
RSA
$EM^d \bmod n$
$|EM^d \bmod n| \approx \log_2 n$,
e.g., $|EM^d \bmod n| = 4096$

Figure 30: RSA PKCS v.1.5

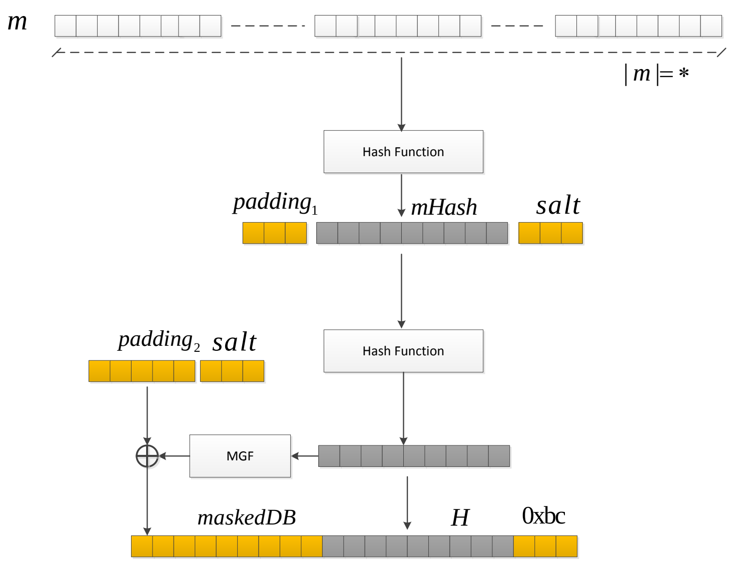


Figure 31: RSA PSS

a key-exchange protocol a reason for which we do not follow the definition pattern for public key encryption schemes (note that we add two steps called: exchange and compute). However, the El-Gamal encryption is directly extractable from this exchange, there is not much difference between the two.

Construction 26. The Diffie-Hellman-Merkle Key exchange. This construction offers a method for exchanging a secret key between two parties.

1. Gen$(1^k)$ (or the key-setup algorithm) fix a prime $p$ and choose generator $g \in Z_p$
2. Exchange is the exchange step between the two parties:

$$A \rightarrow B : g^a \ (a \text{ is a fresh secret random value})$$

$B \rightarrow A : g^b$ ($b$ is a fresh secret random value)

3. Compute is the computational step in which the principals recover the common key. $A$ computes $(g^b)^a$ and B computes $(g^a)^b$, due to commutativity of exponents the common key is $b^{ab}$

**Man-in-the-middle issue**. The Diffie-Hellman protocol is vulnerable to a man-in-the-middle attack, however it is trivial to derive secure versions of it. Moreover, authenticity is not a goal of this protocol (it was intended to exchange securely a key in the sense of secrecy, which it succeeds in doing) therefore this should not be seen as a vulnerability of the main proposal.

**Security considerations**. The order of the group must have a large prime factor, usually one works with $p = 2q + 1$ (primes of this form are usually called safe primes).

***Summary.***

- *Diffie-Hellman provides a method for securely exchanging a key between two parties over an insecure channel (it does not assure authenticity and thus suffers from a man-in-the-middle attack)*

## 1.10 The ELGamal Encryption and Digital Signature

The ElGamal signature scheme was published by Taher ElGamal in 1984 [4]. Discrete logarithms were known to the cryptography community since the 1976 work of Diffie&Hellman, but a signing scheme eluded for many years. We begin with the ElGamal cryptosystem which is a direct result of the Diffie-Hellman key-exchange.

Construction 27. ELGamal Encryption. The ElGamal encryption is a triple of the following algorithms:

1. $\mathsf{Gen}(1^l)$ is the key generation algorithm, it fixes a prime $p$, selects generator $g \in Z_p$, random $a \in Z_p$ and computes $g^a \text{mod} p$. The public key is $\text{Pb} = (g, g^a \text{mod} p, p)$ and the private key is $\text{Pv} = (g, a, p)$.

2. $\mathsf{Enc}(m, e, n)$ is the encryption algorithm. It chooses a random value $k$ and computes $c_1 = g^k \text{mod} p$ then $c_2 = (g^a)^k m \text{mod} p$. The ciphertext is the pair $c = (c_1, c_2)$.

3. $\mathsf{Dec}(c, d, n)$ is the decryption algorithm, the message is recovered as $m = (c_1)^{-a} c_2$.

**Remarks on ElGamal encryption**. The same remark for the order of the group as in the case of Diffie-Hellman holds. When computing $c_2$, multiplication is used to conceal the message, but you can use other operations as well (XOR, symmetric encryption, etc.) that is ElGamal encryption is nothing more but symmetric encryption with the Diffie-Hellman key.

Construction 28. ELGamal Signature. The ElGamal digital signature scheme is a triple of the following algorithms:

1. $\mathsf{Gen}(1^k)$ is the key generation algorithm, it fixes a prime $p$, selects generator $g \in Z_p$, random $a \in Z_p$ and computes $g^a mod p$. The public key is $\text{Pb} = (g, g^a \text{mod} p, p)$ and the private key is $\text{Pv} = (g, a, p)$.

2. $\text{Sig}(m, g, a, p)$ is the signing algorithm, it generates random $k \in (0, p-1)$ computes $r = g^k \text{mod} p$, $s = k^{-1}(h - ar) mod (p-1)$ where $h$ denotes the hash of message $m$.

3. $\text{Ver}(r, s, m, g, n)$ is the verification algorithm, it checks that $r \in (0, p)$, $s \in (0, p-1)$ return 0 if not, otherwise verifies that $g^h = y^r r^s$ and returns 1 if this holds or zero otherwise.

**ElGamal vs. RSA signature**. This note concerns the computational efficiency of ElGamal when compared to RSA. Key generation is cheaper than for RSA (only one prime needed), more, the prime field can be a global parameter, i.e., more entities can use the same fixed $p$. Signing requires more computations but these are done over a prime $p$ that is usually smaller than the RSA modulus, therefore it's faster. Verification is slower than for RSA if special public exponents are used in RSA, i.e., 65537, etc.

**Security equivalence of ElGamal signature**. So far there exist no security reductions (proofs) for ElGamal signatures, nor for DSA that we discuss next. Schnorr's signature is the simplest discrete logarithm based signature to have a security reduction to the underlying problem but is quite absent in practice.

**Relevance of randomness for parameter** $k$. Selecting a random k is mandatory for the security of the ElGamal signature, if k is not random then the secret is trivial to recover as we show next. Let the first signature be

$$r_1 = g^k \text{mod} p, s_1 = k^{-1}(h_1 - ar)\text{mod}(p-1)$$

and the second

$$r_2 = g^k \text{mod} p, s_2 = k^{-1}(h_2 - ar)\text{mod}(p-1),$$

then

$$k = (s_1 - s_2)/(h_1 - h_2).$$

Now secret $a$ can also be recovered from $s_1$ or $s_2$ leading to a total break.

The Digital Signature Algorithm (DSA), also known as the Digital Signature Standard (DSS) is standardized by NIST. It is a variation of the ElGamal signature and all of the previous remarks apply here as well. It differs from ElGamal mostly at the key generation and verification, resulting in smaller signatures (a small but true practical advantage).

Construction 29. Digital Signature Algorithm (DSA). The DSA signature scheme is a triple of the following algorithms:

1. $\mathsf{Gen}(1^k)$ is the key generation algorithm, it fixes a prime $p$ such that another prime $q$ divides $p-1$, selects generator $g \in Z_p$ of order $q$, random $a \in Z_p$ and computes $g^a \text{mod} p$. The public key is $\text{Pb} = (g, g^a \text{mod} p, p)$ and the private key is $\text{Pv} = (g, a, p)$.
2. $\text{Sig}(m, g, a, p)$ is the signing algorithm, it generates random $k \in (0, p-1)$ computes $r = g^k \text{mod} p$, $s = k^{-1}(h + ar) mod(p-1)$ where $h$ denotes the hash of message $m$. The signature is the pair $r, s$.
3. $\text{Ver}(r, s, m, g, n)$ is the verification algorithm, it checks that $r \in (0, p)$, $s \in (0, q)$ returns 0 if not, otherwise verifies that $v = r$ returning 1 if this holds or zero otherwise, where $v = g^{u_1} y^{u_2}$, $u_1 = wh \text{mod} q$, $u_2 = rw \text{mod} q$, $w = s^{-1} \text{mod} q$.

**DSA parameter q**. Parameter q here is fixed at 160 bits according to the output size of SHA1, it can be set to 224 and 256 for SHA2 (see FIPS 186-3).

***Summary.***

- *ElGamal encryption is a direct application of the Diffie-Hellman key exchange*
- *ElGamal signature allows signing based on a trapdoor offered by the difficulty of discrete logarithms (tough equivalence is not yet proven), this construction is by no means trivial to derive from Diffie-Hellman key exchange (several years until its publication prove this)*
- *DSA is a standardize signature scheme, similar in nature to ElGamal*

## 1.11 Computational relations

**Computational problems behind factoring based schemes**. All of the following computational problems nicely reduce one to another: Factoring, Rabin Decryption, RSA Key Generation and computation of Euler Phi. It is just RSA Decryption for which there is no proof that it will allow solving the others, this may mean that RSA Decryption is easier than the rest, on one hand, or simply the proof is not so easy to reach. So far, there is not known algorithm to perform RSA decryption other than using the private key which is equivalent to factoring. In Figure 32 we depict these relations, note that an arrow from P1 to P2 means that if you can solve P1, you can also solve P2.

**Computational problems behind DLog based schemes**. There are three computational problems related to discrete logarithms:

- Decisional Diffie-Hellman problem (DDH), let $y_0 = g^{ab}, y_1 = g^r$ , and $\beta$ a random bit, given $g^a, g^b, y_\beta$ find $\beta$ (that is, distinguish between a complete random value and a DH key),
- Computational Diffie-Hellman problem (CDH) given $g^a, g^b$ compute $g^{ab}$ ,
- Discrete Logarithms (DLog) given $g^a$ compute a.

All of the previous problems assume that you work in a group of finite order, e.g., all operations are done mod$p$. The security of the Diffie-Hellman key exchange is equivalent to CDH (and at most as hard as DLog). If discrete logarithms can be computed then factoring is also easy. In Figure 33 we depict these relations.

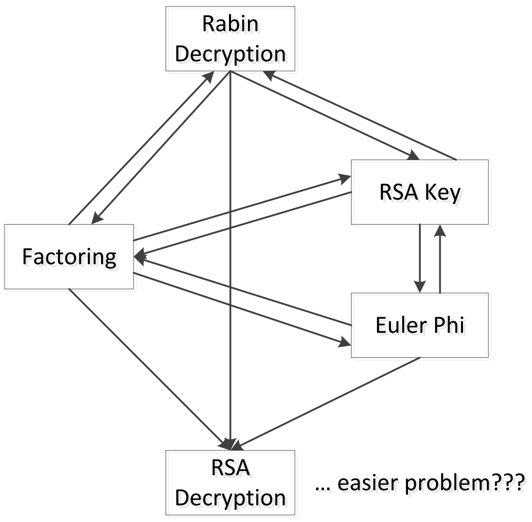


Figure 32: Relations between Factoring and RSA related problem

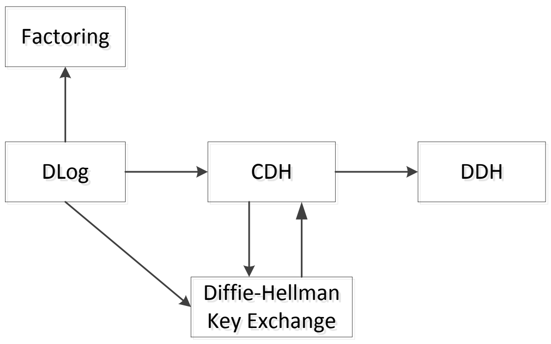


Figure 33: Relations between DLog related problems

***Summary.***

- *inverting the RSA is still unproven to be equivalent to factoring, but there is no better way to break the RSA for now*
- *discrete logs are a harder problem than factoring, in practice this translates to smaller keys (i.e., more efficient cryptosystems)*

## References

[1] M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *Advances in CryptologyASIACRYPT 2000*, pages 531–545. Springer, 2000.

[2] M. Bellare and P. Rogaway. Optimal asymmetric encryption. In *Advances in CryptologyEUROCRYPT'94*, pages 92–111. Springer, 1995.

[3] M. Bellare and P. Rogaway. The exact security of digital signatures-how to sign with rsa and rabin. In *Advances in CryptologyEurocrypt96*, pages 399–416. Springer, 1996.

[4] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology*, pages 10–18. Springer, 1985.

[5] E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. Rsa-oaep is secure under the rsa assumption. In *Advances in CryptologyCRYPTO 2001*, pages 260–274. Springer, 2001.

[6] O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*, volume 2. Cambridge university press, 2009.

[7] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[8] V. Shoup. Oaep reconsidered. In *Advances in CryptologyCRYPTO 2001*, pages 239–259. Springer, 2001.