2

A Brief Introduction to Modern Cryptography

In our Information Age, the need for protecting information is more pronounced than ever. Secure communication for the sensitive information is not only compelling for military or government institutions but also for the business sector and private individuals. The exchange of sensitive information over wired and/or wireless Internet, such as bank transactions, credit card numbers and telecommunication services are already common practices. As the world becomes more connected, the dependency on electronic services has become more pronounced. In order to protect valuable data in computer and communication systems from unauthorized disclosure and modification, reliable non-interceptable means for data storage and transmission must be adopted.

Figure 2.1 shows a hierarchical six-layer model for information security applications. Let us analyze that figure from a top-down point of view. On layer 6, several popular security applications have been listed such as: secure e-mail, digital cash, e-commerce, etc. Those applications depend on the implementation in layer 5 of secure authentication protocols like SSL/TLS, IPSec, IEEE 802.11, etc. However, those protocols cannot be put in place without implementing layer 4, which consists on customary security services such as: authentication, integrity, non-repudiation and confidentiality. The underlying infrastructure for such security services is supported by the two pair of cryptographic primitives depicted in layer 3, namely, encryption/decryption and digital signature/verification. Both pair of cryptographic primitives can be implemented by the combination of public-key and private key cryptographic algorithms, such as the ones listed in layer 2. Finally, in order to obtain a high performance from the cryptographic algorithms of layer 1, it is indispensable to have an efficient implementation of arithmetic operations such as, addition, subtraction, multiplication, exponentiation, etc.

In the rest of this Chapter we give a short introduction to the algorithms and security services listed in layers 2-4. Hence, the basic concepts of cryptography, fundamental operations in cryptographic algorithms and some im-

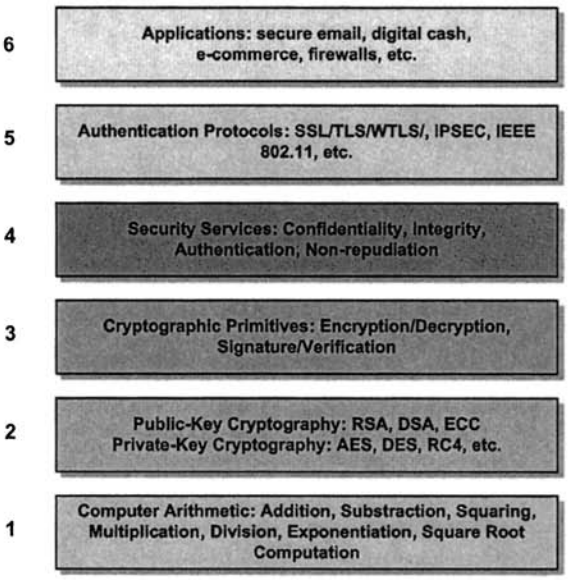

Fig. 2.1. A Hierarchical Six-Layer Model for Information Security Applications

portant cryptographic applications in the industry are studied and analyzed. Furthermore, alternatives for the implementation of cryptographic algorithms on various software and hardware platforms are also discussed.

2.1 Introduction

A cryptographic cipher system can hide the actual contents of every message by transforming (enciphering) it before transmission or storage. The techniques needed to protect data belong to the field of cryptography, which can be defined as follows.

Definition 2.1. *We define* ***Cryptography*** *as the discipline that studies the mathematical techniques related to Information security such as providing the security services of confidentiality, data integrity, authentication and non-repudiation.*

In the wide sense, cryptography addresses any situation in which one wishes to limit the effects of dishonest users [110]. Security services, which include confidentiality, data integrity, entity authentication, and data origin authentication [228], are defined below.

- **Confidentiality:** It guarantees that the sensitive information can only be accessed by those users/entities authorized to unveil it. When two or more parties are involved in a communication, the purpose of confidentiality is to guarantee that only those two parties can understand the data exchanged. Confidentiality is enforced by encryption.
- **Data integrity:** It is a service which addresses the unauthorized alteration of data. This property refers to data that has not been changed, destroyed, or lost in a malicious or accidental manner.
- **Authentication:** It is a service related to identification. This function applies to both entities and information itself. Two parties entering into a communication should identify each other. Information delivered over a channel should be authenticated as to origin, date of origin, data content, time sent, etc. For these reasons this aspect of cryptography is usually subdivided into two major classes: *entity authentication* and *data origin authentication*. Data origin authentication implicitly provides data integrity.
- **Non-repudiation:** It is a service which prevents an entity from denying previous commitments or actions. For example, one entity may authorize the purchase of property by another entity and later deny such authorization was granted. A procedure involving a trusted third party is needed to resolve the dispute.

In cryptographic terminology, the message is called *plaintext*. Encoding the contents of the message in such a way that its contents cannot be unveiled by outsiders is called *encryption*. The encrypted message is called the *ciphertext*. The process of retrieving the plaintext from the ciphertext is called *decryption*. Encryption and decryption usually make use of a *key*, and the coding method use this key for both encryption and decryption. Once the plaintext is coded using that key then the decryption can be performed only by knowing the proper key.

Cryptography falls into two important categories: secret and public key cryptography. Both categories play their vital role in modern cryptographic applications. For several crucial applications, a combination of both secret and public key methods is indispensable.

2.2 Secret Key Cryptography

Definition 2.2. *Matematically, a* **symmetric key cryptosystem** *can be defined as the tuple* $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$*, where [110]:*

$\mathcal{P}$ *represents the set of finitely many possible plain-texts.*
$\mathcal{C}$ *represents the set of finitely many possible cipher-texts.*
$\mathcal{K}$ *represents the key space, i.e, the set of finitely many possible keys.*
$\forall K \in \mathcal{K}\ \exists E_K \in \mathcal{E}$ *(encryption rule),* $\exists D_K \in \mathcal{D}$ *(decryption rule).*
Each $E_K : \mathcal{P} \rightarrow \mathcal{C}$ *and* $D_K : \mathcal{C} \rightarrow \mathcal{P}$ *are well-defined functions such that* $\forall x \in \mathcal{P}, D_K(E_K(x)) = x$.

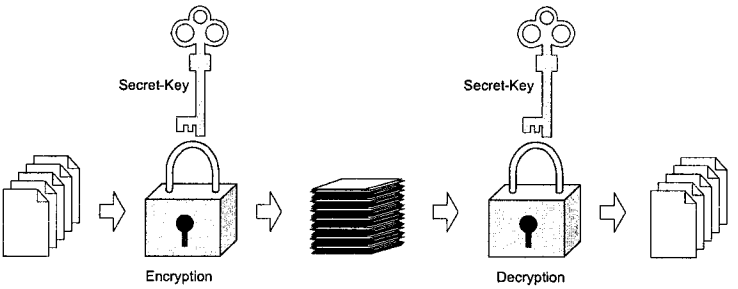

Fig. 2.2. Secret Key Cryptography

Both encryption and decryption keys (which sometimes are the same keys) are kept secret and must be known at both ends to perform encryption or decryption as is shown in Fig. 2.2. Symmetric algorithms are fast and are used for encrypting/decrypting high volume data. It is customary to classify symmetric algorithms into two types: stream ciphers and block ciphers.

- **Stream ciphers:** A stream cipher is a type of symmetric encryption algorithms in which the input data is encrypted one bit (sometimes one byte) at a time. They are sometimes called state ciphers since the encryption of a bit is dependent on the current state. Some examples of stream ciphers are SEAL, TWOPRIME, WAKE, RC4, A5, etc.
- **Block ciphers:** A block cipher takes as an input a fixed-length block (plaintext) and transform it into another block of the same length (ciphertext) under the action of a user-provided secret key. Decryption is performed by applying the reverse transformation to the ciphertext block using the same secret key. Modern block ciphers typically use a block length of 128 bits. Some famous block ciphers are DES, AES, Serpent, RC6, MARS, IDEA, Twofish, etc.

The most popular block cipher algorithm used in practice is DEA (*Data Encryption Algorithm*) defined in the standard DES [251]. The secret key used in DEA has a bit-length of 56 bits. Even though that key length was considered safe back in the middle 70's, nowadays technology can break DEA in some few hours by launching a brute-force attack. That is why DEA is widely used as Triple DEA (TDEA) which may offer a security equivalent to 112 bits. TDEA uses three 56-bit keys (namely, K_1, K_2 and K_3). If each of these keys is independently generated, then this is called the three key TDEA (3TDEA). However, if K_1 and K_2 are independently generated, and K_3 is set equal to K_1, then this is called the two key TDEA (2TDEA) [258].

On October 2000, a new symmetric cryptographic algorithm "Rijndael" was chosen as the new Advanced Encryption Standard (AES) [60] by NIST (National Institute of Standards and Technology) [253]. Due to its enhanced

security level, it is replacing DEA and triple DEA (TDEA) in a wide range of applications.

Although all aforementioned secret key ciphers offer a high security and computational efficiency, they also exhibit several drawbacks:

- **Key distribution and key exchange** The master key used in this kind of cryptosystems must be known by the sender and receiver only. Hence, both parties should prevent that this key can get compromised by unauthorized entities[1].
- **Key management** Those system having many users, must generate/manage many keys. For security reasons, a given key should be changed frequently, even in every session.
- **Incompleteness** It is impossible to implement some of the security services mentioned before. In particular, *Authentication* and *non-repudiation* cannot be fully implemented by only using secret key cryptography [317].

2.3 Hash Functions

Definition 2.3. *A Hash function H is a computationally efficient function that maps fixed binary chains of arbitrary length $\{0,1\}^*$ to bit sequences $H(B)$ of fixed length. $H(B)$ is the hash value or digest of B.*

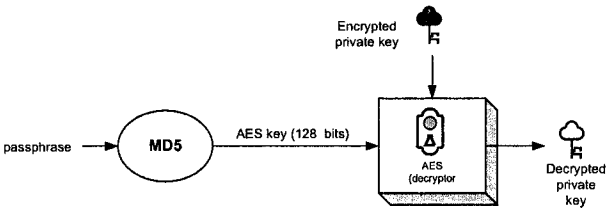

Fig. 2.3. Recovering Initiator's Private Key

In words, a hash function h maps bit-strings of arbitrary finite length to strings of fixed length, say n bits. MD5 and SHA-1 are two examples of hash functions. MD5 produces 128-bit hash values while SHA-1 produces 160-bit hash values.

Hash functions can be used for protecting user's secret key as depicted in Fig. 2.3. Fig. 2.3 shows the customary procedure used for accomplishing that

[1] This implies that in a community of n users a total of $\frac{n(n-1)}{2}$ secret keys must be created so that all users can communicate with each other in a confidential manner.

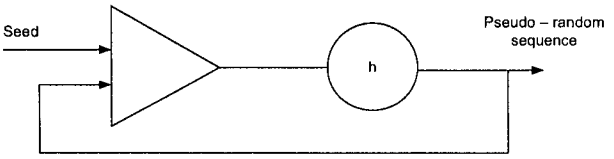

Fig. 2.4. Generating a Pseudorandom Sequence

goal. It is noticed that the AES secret key is generated by means of the hash value corresponding to the pass-phrase given by the user. Another typical application of Hash functions is in the domain of pseudorandom sequences as shown in Fig. 2.4.

Nevertheless, the main application of hash function is as a key building block for generating digital signatures as it is explained in the next Section.

2.4 Public Key Cryptography

A breakthrough in Cryptography occurred in 1976 with the invention of *public key cryptography* by Diffie and Hellman[2] [68]. This invention not only solved the key distribution and management problem but also it provided the necessary tool for implementing authentication and non-repudiation security services effectively.

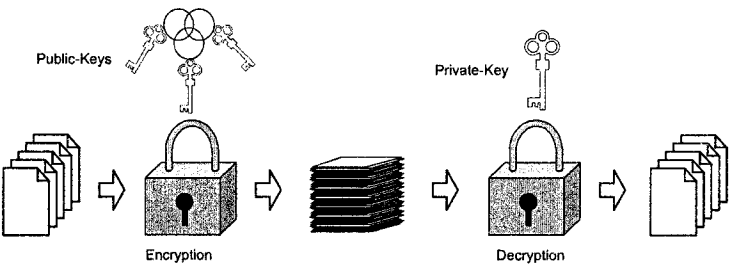

Fig. 2.5. Public Key Cryptography

[2] Although Diffie and Hellman were the first in publishing the concepts of public key cryptography in the open literature, we know now that they were not the first inventors. In 1997, a British Security agency (CESG, *National Technical Authority for Information Assurance*) published documents showing that in fact James Ellis and Clifford Cocks came out with the mechanisms needed for performing RSA-like public key cryptography in 1973. Short after that, M. Williamson discovered what is now known as Diffie-Hellman key exchange [374, 317, 206].

Asymmetric algorithms use a different key for encryption and decryption, and the decryption key cannot be easily derived from the encryption key. Asymmetric algorithms use two keys known as public and private keys as shown in Fig. 2.5.

The public key is available to everyone at the sending end. However a private or secret key is known only to the recipient of the message. An important characteristic of any public key system is that the public and private keys are related in such a way that only the public key can be used to encrypt (decrypt) messages and only the corresponding private key can be used to decrypt(encrypt) them.

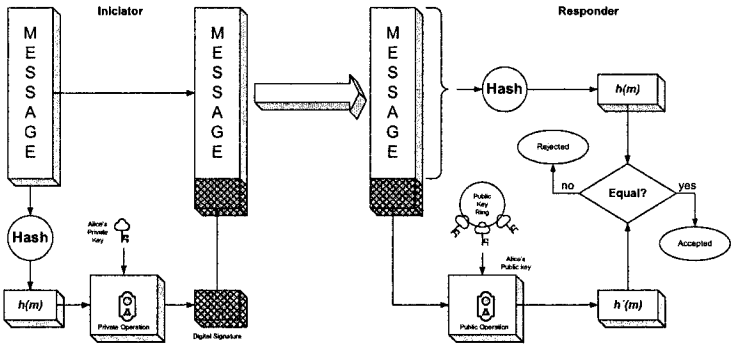

Fig. 2.6. Basic Digital Signature/Verification Scheme

Public key cryptosystems can be used for generating *digital signatures*, which cannot be repudiated. The concept of digital signature is analog to the real-world autograph signature, but it is more powerful as it also protects against malicious data modifications. A digital signature scheme is based in two algorithms: signature and verification as explained below.

- A encrypts the message m using its private key $c_1 := E_{K_{priv}(A)}(m)$
- A encrypts the result c_1 using B's public key and send the result to B,

$$c = E_{K_{pub}(B)}(c_1) = E_{K_{pub}(B)}\{E_{K_{priv}(A)}(m)\}$$

- B recovers m by performing,

$$m = D_{K_{pub}(A)}\{D_{K_{priv}(B)}(c)\}$$

Since B is able to recover m using A's public key, B can verify whether A really sign the message using its private key. Moreover, since the signature depends on the message contents, theoretically nobody else can reuse the same signature in any other message.

In practice, as is shown in Fig.2.6, a digital signature is applied not to the document to be signed itself, but to its *hash* value. This is due to efficiency reasons as public key cryptosystems tend to be computationally intensive. A hash function H is applied to the message to append its hash value $h = H(M)$, to the document itself. Thereafter, h is signed by "encrypting" it with the private key of the sender. This becomes the signature part of the message.

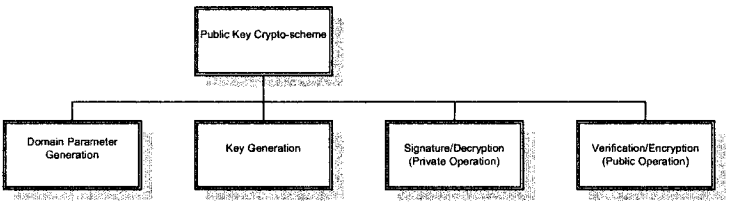

Fig. 2.7. Public key cryptography Main Primitives

As shown in Fig. 2.7 Public key cryptosystems' main primitives are:

1. **Domain Parameter Generation.** This primitive creates the mathematical infrastructure required by the particular cryptosystem to be used.
2. **Key Generation.** This primitive create users' public/private key.
3. **Public Operation.** This primitive is used for encrypting and/or verifying messages.
4. **Private Operation.** This primitive is used for decrypting and/or signing messages.

Theoretically, a public key cryptosystem can be constructed by means of specialized mathematical functions called "trapdoor one-way functions" which can be formally defined as follows.

Definition 2.4. *A One-way Function [110] is an injective function* $f(x)$

$$f : \{0,1\}^* \rightarrow \{0,1\}^*,$$

such that $f(x)$ *can be computed efficiently, but the computation of* $f^{-1}(y)$ *is computational intractable, even when using the most advanced algorithms along with the most sophisticated computer systems. We say that a one-way function is a One-way trapdoor function if is feasible to compute* $f^{-1}(y)$ *if and only if a supplementary information (usually the secret key) is provided.*

In words, a *one-way* function f is easy to compute for any domain value x, but the computation of $f^{-1}(x)$ should be computationally intractable. A trapdoor one-way function is a one-way function such that the computation $f^{-1}(x)$ is easy, provided that certain special additional information is known. The following three problems are considered among the most common for creating *trapdoor one-way* functions.

- ***Integer Factorization problem:*** Given an integer number n, obtain its prime factorization, i.e., find $n = {p_1}^{e_1}{p_2}^{e_2}{p_3}^{e_3}\cdots{p_k}^{e_k}$, where p_i is a prime number and $e_i \geq 1$.
  It is noticed that finding large prime numbers[3] is a relatively easy task, but solving the problem of factorizing the product of prime numbers is considered computationally intractable if the prime numbers are chosen carefully and with a sufficient large bit-length [196].
- ***Discrete Logarithm problem:*** Given a number p, a generator $g \in {\mathbb{Z}_p}^*$ and an arbitrary element $a \in {\mathbb{Z}_p}^*$, find the unique number i, $0 \leq i < p-1$, such that $a \equiv g^i (mod\, p)$.
  This problem is useful in cryptography due to the fact that finding discrete logarithms is difficult. The brute-force method for finding $g^j (mod\, p)$ for $1 < j \leq p-1$ is computationally unfeasible for sufficiently large prime values. However, the field exponentiation operation can be computed efficiently. Hence, $g^i (mod\, p)$ can be seen as a *trapdoor one-way function* function for certain values of p.
- ***Elliptic curve discrete Logarithm problem:*** Let $E_{\mathbb{F}_q}$ be an elliptic curve defined over the finite field $F_q$and let P be a point $P \in E_{\mathbb{F}_q}$ with primer order n. Consider the k-multiple of the point P, $Q = kP$ defined as the elliptic curve point resulting of adding P, $k-1$ times with itself, where k is a positive scalar in $[\![1, n-1]\!]$. The elliptic curve discrete logarithm problem consists on finding the scalar k that satisfies the equation $Q = kP$. This problem is considered a strong one-way trapdoor function due to the fact that computing k given Q and P is a difficult computational problem. However, given k is relatively easy to obtain the k-th multiple of P, namely, $Q = kP$.

2.5 Digital Signature Schemes

- $\mathcal{M}$ represents the set of all finitely many messages that can be signed
- $\mathcal{S}$ represents the set of all finitely many signatures (usually the signatures are fixed-length binary chains).
- $\mathcal{K}_{\mathcal{S}}$ represents the set of private keys.
- $\mathcal{K}_{\mathcal{V}}$ represents the set of public keys.
- $S_{\mathcal{E}} \colon \mathcal{M} \longrightarrow \mathcal{S}$ represents the transformation rule for an entity $\mathcal{E}$.
- $V_{\mathcal{E}} \colon \mathcal{M} \times \mathcal{S} \longrightarrow \{true, false\}$ represents the verification transformation for signatures produced by $\mathcal{E}$. It is used for other entities in order to verify signatures produced by $\mathcal{E}$.

  $S_{\mathcal{E}}$ y $V_{\mathcal{E}}$ define a digital signature scheme for $\mathcal{E}$.

Definition 2.5. *A Digital signature scheme is the triple* (Gen, Sig, Ver) *of algorithms such that,*

[3] In the cryptography domain a large prime number has a bit-length of at least 512 bits.

i. Gen is a Key generation algorithm, with input s; known as the security parameter; and possibly another extra information I, which gives as an ouptut $(\mathbf{k}_{\mathcal{S}}, \mathbf{k}_{\mathcal{V}}) \in \mathcal{K}_{\mathcal{S}} \times \mathcal{K}_{\mathcal{V}}$ *corresponding to private key, and public key, respectively.*

ii. Sig is a Signature algorithm, with input $(\mathbf{m}, \mathbf{k}_{\mathcal{S}}) \in \mathcal{M} \times \mathcal{K}_{\mathcal{S}}$*, which gives as an output an element* $\sigma \in \mathcal{S}$*, called Signature (of the message* $\mathbf{m}$ *with the private key* $\mathbf{k}_{\mathcal{S}}$*).*

iii. Ver is a Verification algorithm, with input $(\mathbf{m}, \sigma, \mathbf{k}_{\mathcal{V}}) \in \mathcal{M} \times \mathcal{S} \times \mathcal{K}_{\mathcal{V}}$*, which gives as an output the set* $\{true, false\}$ *and*

$$Ver(\mathbf{m}, Sig(\mathbf{m}, \mathbf{k}_{\mathcal{S}}), \mathbf{k}_{\mathcal{V}}) = true$$

$\forall$ *valid* $(\mathbf{k}_{\mathcal{S}}, \mathbf{k}_{\mathcal{V}})$ *obtained from Gen and for all* $\mathbf{m} \in \mathcal{M}$.

Undoubtedly, the most popular public-key algorithms are RSA (based on factoring large numbers), DSA and ElGamal (based on discrete log problem) and Elliptic Curve Cryptosystems. Elliptic curve cryptography is now popular due to the fact that it offers the same security level as offered by other contemporary algorithms at a shorter key length. It is based on elliptic curve addition operation.

2.5.1 RSA Digital Signature

The most popular algorithm for commercial applications is RSA[4]. RSA algorithm is symmetric in the sense that both, the public key and the private key can be utilized for encrypting a message.

RSA Key Generation

Algorithm 2.1 shows RSA key generation procedure. The public key is composed by the two integers (n, e), where n is called the RSA modulus and is defined as the product of two prime numbers p, q, of approximately the same bit-length. Both, p, q should be generated randomly and must be kept secret. The number e is called the *public exponent.* It must satisfy: $1 < e < \phi$ and $\gcd(e, \phi) = 1$ where $\phi = (p-1)(q-1)$. The private key d is called the *private exponent* and it must satisfy: $1 < d < \phi$ and $ed \equiv 1 (\text{mod } \phi)$. It is noticed that the problem of determining the key d given the public key (n, e) has a computacional difficulty equivalent to the integer factorization problem of finding p or q given n.

[4] RSA stands for the first letter in each of its inventors' last names: Rivest, Shamir and Addleman. These three distinguished professors were declared the 2002 A.M Turin award winners. At that time, Professor Shamir consider it "the ultimate seal of approval" for Cryptography as a Computer Science discipline [325].

Algorithm 2.1 RSA Key Generation

Require: bit-length k, a public exponent e, where e is a small prime number.
Ensure: RSA public key (n, e) and private key d.
1: Randomly find two primes $\frac{k}{2}$-bit numbers p and q.
2: $n = pq$;
3: $\phi(n) = (p-1)(q-1)$;
4: **if** $\gcd(e, \phi(n)) \neq 1$ **then**
5: Go to Step 1.
6: **end if**
7: Find d such that $d = e^{-1} \bmod \phi(n)$.
8: **Return** (n, e, d).

RSA Digital Signature

RSA encryption/decryption and Signature/verification are based in the Euler theorem identity, which establishes that,

$$m^{ed} = m \pmod{n} \tag{2.1}$$

for any arbitrary integer m. Signature and verification processes are shown in Algorithms 2.2 and 2.3. The author A of the message m computes the hash value $h = H(m)$. Then, A computes the signature $s = h^d$. Then A can send the message m along with the signature s to a verifying entity, say B. B can verify A's signature as follows. It recovers the hash value from s by computing $\hat{h} = s^e$. Thereafter, B computes once again the hash value, say, $h = H(m)$. If $\hat{h} = h$, then the signature is accepted otherwise, it is rejected.

Algorithm 2.2 RSA Digital Signature

Require: Sender's public key (n, e), Sender's private key d, message m.
Ensure: digital signature s.
1: $h = H(m)$;
2: $s = h^d \bmod n$.
3: **Return** s.

2.5.2 RSA Standards

RSA is specified in [193, 253, 255]. Additionally, there exist a number of standards where the digital signature algorithm RSA just described is utilized. The Public Key Cryptography Standard (PKCS), is a set of standards that include among others, PKCS#1[5], PKCS#3[6] and PKCS#12[7]. PKCS series

[5] RSA Cryptography Standard
[6] Diffie-Hellman key agreement Standard
[7] Personal Information Exchange Syntax Standard

Algorithm 2.3 RSA Signature Verification

Require: Sender's public key (n, e), message m, digital signature s.
Ensure: Accept/Reject.
1: $h = H(m)$;
2: $\hat{h} = s^e \bmod n$;
3: **if** $h = \hat{h}$ **then**
4: **Return**(Accept);
5: **else**
6: **Return**(Reject);
7: **end if**

have become part of many formal and de facto standards, including ANSI X9 documents, PKIX, SET, S/MIME, and SSL [193].

2.5.3 DSA Digital Signature

The Digital Signature Algorithm (DSA) is based in the crypto-scheme proponed by ElGamal in 1984, which in turn is based on the discrete logarithm problem. Many versions of the original ElGamal procedure has been proposed. In 1991, the ElGamal procedure was adopted by the U.S. National Institute of Standards and Technology and registered under the name of Digital Signature Standard (DSS).

DSA Key Generation

The prime numbers p and q and the generator g are public domain parameters. They define a multiplicative *Abelian group* modulus p. The parameter $g \in [2, p-1]$ specifies a generator of the multiplicative cyclic subgroup $\langle g \rangle$ of order q. This mathematically implies that $q|(p-1)$ and no other smaller positive integer is a prime divisor of $p-1$ satisfying $g^q \equiv 1$. The private key x is randomly selected among the subgroup elements, i.e., $x \in [1, q-1]$, whereas the corresponding public key is generated by computing $y = g^x \bmod p$, as is shown in Algorithm 2.5. The problem of finding x given the domain parameters (p, q, g) and the public key y is known as the *discrete logarithm problem.*

DSA Digital Signature Algorithm

Once that the public/private key pair has been generated, a given entity A can generate the DSA signature $S = (r, s)$ of a message m by proceeding as follows (see Algorithm 2.6). First, A must select a random number $k \in [1, q-1]$, which must be secret and should be destroyed after the DSA has been generated. Then, A must compute $T = g^k \bmod p$, and $r = T \bmod q$. Thereafter, the message m is processed using a secure hash algorithm H so that $h = H(m)$ is

Algorithm 2.4 DSA Domain Parameter Generation

Require: Security parameters l and t.
Ensure: Domain parameters (p, q, g).
1: Select a prime number q of t bits and another prime number p of l bits such that $q|(p-1)$.
2: Find an element g of order q.
3: **repeat**
4: randomly select a number $h \in [1, p-1]$ and compute $g = h^{\frac{p-1}{q}} \mod p$.
5: **until** $\{g \neq 1\}$
6: **Return** (p, q, g).

Algorithm 2.5 DSA Key Generation

Require: Domain parameters p, q, g.
Ensure: Private key x and public key y.
1: Randomly select $x \in [1, q-1]$.
2: $y = g^x \mod p$;
3: **Return** (y, x).

computed. Then, the other component of the DSA signature can be computed as,

$$s \equiv k^{-1}(h + xr) \mod q \tag{2.2}$$

DSA signature is composed by the pair (s, r). The verifying entity B can check the correctness of the DSA based on the following observation,

$$k \equiv s^{-1}(h + xr) \mod q. \tag{2.3}$$

Which implies,

$$g^k \equiv g^{s^{-1}h} g^{xs^{-1}r} \mod p \tag{2.4}$$

Finally, knowing that $T = g^k \mod p$ and $y = g^x \mod p$, we have,

$$T \equiv g^{hs^{-1}} y^{rs^{-1}} \mod p \tag{2.5}$$

Lats equation corresponds to the computation accomplished by the verifier at line 8 of Algorithm 2.7. Therefore, the verifier entity B can assess the correctness of a DSA signature by verifying that the equality $r = T \mod q$ holds. This can be done by knowing the domain parameters (p, q, g), the public key y and the DSA signature (r, s). DSA signature generation and verification are shown in Algorithms 2.6 and 2.7, respectively.

2.5.4 Digital Signature with Elliptic Curves

Elliptic curves over real numbers are defined as the set of points (x, y) which satisfy the elliptic curve equation of the form:

$$y^2 = x^3 + ax + b \tag{2.6}$$

Algorithm 2.6 DSA Signature Generation

Require: domain parameters (p, q, g), Sender's private key x, message m.
Ensure: Signature (r, s).
1: randomly select $k \in [1, q-1]$.
2: $T = g^k \bmod p$;
3: $r = T \bmod q$;
4: **if** $r = 0$ **then**
5: Go to Step 1;
6: **end if**
7: $h = H(m)$;
8: $s = k^{-1}(h + xr) \bmod q$;
9: **if** $s = 0$ **then**
10: Go to Step 1;
11: **end if**
12: **Return** (r, s).

Algorithm 2.7 DSA Signature Verification

Require: Domain parameters (p, q, g), Sender's public key y, message m and signature (r, s).
Ensure: Accept/Reject.
1: **if** r, s are not in the interval $[1, q-1]$ **then**
2: **Return**("Reject")
3: **end if**
4: $h = H(m)$;
5: $w = s^{-1} \bmod q$;
6: $u_1 = hw \bmod q$;
7: $u_2 = rw \bmod q$;
8: $T = g^{u_1} y^{u_2} \bmod p$;
9: $\hat{r} = T \bmod q$;
10: **if** $r = \hat{r}$ **then**
11: **Return**(Accept);
12: **else**
13: **Return**(Reject);
14: **end if**

$$y^2 = x^3 + ax + b \tag{2.6}$$

where a and b are real numbers. Each choice of a and b produces a different elliptic curve as shown in Figure 4.1. The elliptic curve in Equation 2.6 forms a group if $4a^3 + 27b^2 \neq 0$. An elliptic curve group over real numbers consists of the points on the corresponding elliptic curve, together with a special point $\mathcal{O}$ called the point at infinity. Elliptic curve groups are additive groups; that is, their basic function is addition. The negative of a point $P = (x, y)$ is its reflection in the x-axis: the point $-P$ is $(x, -y)$. If the point P is on the curve, the point $-P$ is also on the curve.

In elliptic curve cryptography we are only interested in elliptic curves defined over finite fields. This means that the coordinates of the points in the elliptic curve can only take values that belong to the finite field over which, the elliptic curve has been defined. In particular we define elliptic curves over binary extension fields $GF(2^m)$, using the following adjusted curve equation,

$$y^2 + xy = x^3 + ax^2 + b \tag{2.7}$$

where $a, b \in GF(2^m)$ and $b \neq 0$. Once again, the elliptic curve includes all the points (x, y) that satisfy above equation in $GF(2^m)$ arithmetic, plus the point at infinity $\mathcal{O}$. The set of point that belong to the curve E is denoted as $E(\mathbb{F}_{2^m})$[8].

Elliptic Curve Domain Parameters

The *domain parameters* needed for obtaining a public key cryptosystem based on the elliptic curve discrete logarithm problem over $\mathbb{F}_q$ are the following [133],

1. The number of field elements (finite field order) q.
2. The coeffcients $a, b \in \mathbb{F}_q$ that define the elliptic equation E over $\mathbb{F}_q$.
3. A base point $P = (x_p, y_p) \in \mathbb{F}_q$ that belongs to the curve E. P must have a prime order.
4. The *order* n of P.
5. The *cofactor* $h = \#E(\mathbb{F}_q)/n$.

ECDSA Key Generation

Let $P \in E(\mathbb{F}_q)$ with order n, where E is an elliptic curve as defined above. We consider the field order q, the elliptic curve equation E and the base point P as public domain parameters. The private key d is a randomly chosen integer in the range $[1, n-1]$ and the corresponding public key is the point $Q = dP$ as computed in Algorithm 2.8 below. The problem of defining d given P and Q is known as the *elliptic curve discrete logarithm problem.*

Algorithm 2.8 ECDSA Key Generation

Require: Elliptic curve public domain parameters (q, E, P, n).
Ensure: public/private key pair $Q = (x_Q, y_Q)$ and d.
1: Randomly choose d in the range $[1, n-1]$
2: $Q = dP$;
3: **Return** (Q, d).

[8] Elliptic curve theory is covered in Chapter 4. Reconfigurable hardware implementations of elliptic curve cryptosystems are studied in Chapter 10.

ECDSA Digital Signature

Elliptic Curve Digital Signature Algorithm (ECDSA) is the elliptic curve analogue of the Digital Signature Algorithm (DSA) [141]. It was accepted in 1999 as an ANSI standard, and in 2000 it was accepted as IEEE and NIST standards. Unlike the ordinary discrete logarithm problem and the integer factorization problem, no subexponential-time algorithm is known for the elliptic curve discrete logarithm problem. For this reason, the strength-per-key-bit is substantially greater in an algorithm that uses elliptic curves.

Algorithm 2.9 ECDSA Digital Signature Generation

Require: Domain parameters: (q, a, b, P, n, h), Sender's private key d, message m.
Ensure: Signature (r, s).
1: Randomly Select k in the interval $[1, n-1]$
2: $kP = (x_1, y_1)$; and convert x_1 into an integer $\overline{x}_1$.
3: Compute $r = \overline{x}_1 \bmod n$.
4: **if** $r = 0$ **then**
5: goto step 1;
6: **end if**
7: $e = H(m)$;
8: $s = k^{-1}(e + dr) \bmod n$.
9: **if** $s = 0$ **then**
10: goto step 1;
11: **end if**
12: **Return**(r, s).

The ECDSA digital signature algorithm is shown in Fig. 2.9. The signature for this message is the pair (r, s). It is to be noted that the signature depends on the private key and the message. This implies that, at least in theory, no one can substitute a different message for the same signature. Note that if a message m has a valid digital signature (r, s) then,

$$s \equiv k^{-1}(e + dr) \bmod n.$$

which implies,

$$k \equiv s^{-1}(e + dr) \equiv s^{-1}e + s^{-1}dr \equiv we + wdr \equiv u_1 + u_2 \cdot d \bmod n.$$

Thus, $X = u_1P + u_2Q = (u_1 + u_2d)P = kP$, and consequently we validate the signature iff $v = r$. Above verification process is carried out by the procedure shown in Algorithm 2.10. Notice that in line 8 of that procedure, the elliptic curve point $X = u_1 \cdot P + u_2 \cdot Q$, is computed. As explained above, if the signature to be verified is a valid one then the equality $v = \overline{x}_1 \bmod n \equiv r$ should hold.

Algorithm 2.10 ECDSA Signature Verification

Require: Domain parameters: (q, a, b, P, n, h), signature (r, s), Sender's public key Q, message m.
Ensure: Reject/Accept.
1: **if** r, s are not in the interval $[1, n-1]$ **then**
2: **Return**("Reject")
3: **end if**
4: $e = H(m)$;
5: $w = s^{-1} \bmod n$;
6: $u_1 = ew \bmod n$;
7: $u_2 = rw \bmod n$;
8: $X = u_1 \cdot P + u_2 \cdot Q$;
9: **if** $X = \mathcal{O}$ **then**
10: **Return** "Rejected".
11: **end if**
12: Convert the x coordinate of X to an integer $\overline{x}_1$.
13: $v = \overline{x}_1 \bmod n$;
14: **if** $v = r$ **then**
15: **Return**(Accept);
16: **else**
17: **Return**(Reject);
18: **end if**

2.5.5 Key Exchange

In secret key cryptography, it is necessary that both parties at the sending and receiving ends agree on a secret key for transferring data in a secure way. Thus, several key agreement protocols have been proposed in order to establish a shared secret. The first such protocol is the Diffie-Hellman protocol, which provides the key establishment of a key with two message transfers. In the following, we will describe the basic Diffie-Hellman exchange protocol followed by its elliptic curve version.

Diffie-Hellman Key Exchange Protocol

Diffie-Hellman key exchange was invented in 1976 by Whitfield Diffie, Martin Hellman and Ralph Merkle. It was the first practical method for establishing a shared secret over an unprotected communication channel. Let us suppose that A and B have already agreed on working with a group G (for example, let us say the group of integers modulo p) and a generator element g in G. Then, the protocol dataflow is as follows (Figure 2.8):

- A picks a random natural number a and sends g^a to B.
- B picks a random number b and sends g^b to A.
- A computes K=$(g^b)^a$.
- B computes K=$(g^a)^b$.

In the Diffie-Hellman protocol, g and p are the domain parameters and K is the private key for the session which can be used as a shared secret for secure communication between A and B via symmetric cryptography.

Diffie-Hellman protocol is considered secure if G and g are chosen properly, i.e., the eavesdropper has an enormous difficulty to compute the element g^{ab}, because he/she needs to solve the discrete logarithm problem over the group G.

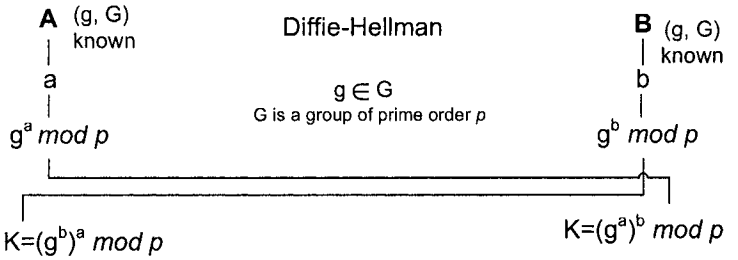

Fig. 2.8. Diffie-Hellman Key Exchange Protocol

Elliptic Curve Diffie-Hellman Key Exchange Protocol

Let A and B agree on an elliptic curve E over a large finite field F and a point P on that curve. Then the necessary steps for exchanging a secret key by using elliptic curve discrete logarithmic algorithm are as shown in Figure 2.9.

- A and B each privately choose large random integers, denoted r_1 and r_2.
- Using elliptic curve point-addition, A computes r_1P on E and sends it to B. B computes r_2P on E and sends it to A.
- Both A and B can now compute the point r_1r_2P by performing the elliptic curve scalar multiplication of the received value of r_2P, r_1P by his/her secret number r_1, r_2, respectively.

A and B agree that the x coordinate of this point will be their shared secret value.

2.6 A Comparison of Public Key Cryptosystems

Due to the high difficulty of computing the elliptic curve discrete logarithm problem, one can obtain the same security provided by other existing public-key cryptosystems, but at the price of much smaller fields, which automatically implies shorter key lengths. Having shorter key lengths means smaller

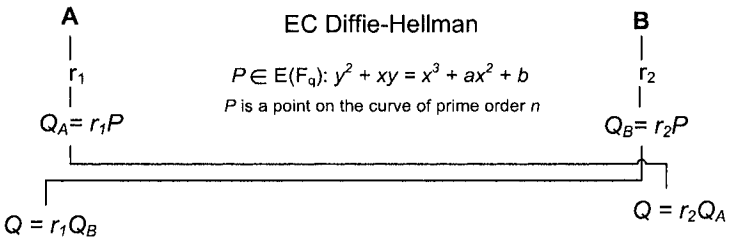

Fig. 2.9. Elliptic Curve Variant of the Diffie-Hellman Protocol

bandwidth and memory requirements. These characteristics are especially important in most embedded system applications, where both memory and processing power are constrained.

High performance implementations of elliptic curve cryptography depend heavily on the efficiency in the computation of the finite field arithmetic operations needed for the elliptic curve operations. On the other hand, the level of security offered by protocols such as the Diffie-Hellman key exchange algorithm relies on exponentiation in a large group. Typically, the implementation of this protocol requires a large number of exponentiation computations in relatively big fields. Therefore, hardware/software implementations of the group operations are, for all the practical sizes of the group, computationally intensive.

Nowadays, there exist algorithms able to solve the factorization problem as well as the discrete logarithm problems in a sub-exponential time. For instance, the Number Field Sieve (NFS) [203] is the best option for solving the integer factorization problem. The Number Field Sieve (NFS) [115] and the Pollard's rho algorithm [273] can solve the discrete logarithm problem.

In the case of RSA, the largest RSA modulus factored is a 640-bit (193-digit) integer in November, 2005 [195]. In the case of ECDSA, the largest known example was solved using the Pollard's rho method for both, prime and binary finite fields. The elliptic curve discrete logarithm problem for an elliptic curve over a 109-bit prime field was broken on November 2002 [44], whereas another elliptic curve defined over a 109-bit binary field was broken in April, 2004. The effort required 2600 computers and took 17 months [45].

2.7 Cryptographic Security Strength

Some of the major factors that determine the *security strength* of a given symmetric block cipher algorithm include, the quality of the algorithm itself, the key size used and the block size handled by the algorithm[9].

The security strength of an n-bit key symmetric block cipher algorithm, which has no known security flaws, is measured in terms of the amount of work it takes to try all possible keys, an attack traditionally known as the *brute-force* attack.

A generic cryptographic algorithm that has an m-bit key, but whose strength is comparable to an n-bit key of a strong symmetric block cipher algorithm is said to have an equivalent n-bit security strength. In general, however, the equivalent n-bit security strength of a given algorithm is less than m due to the possibility that certain specific attack to that algorithm may provide computational advantages compared with the brute-force attack [257].

Determining the security strength of an algorithm is not trivial. For example, one might expect that 3TDEA would provide $56 * 3 = 168$ bits of strength. However, the so-called *birthday and meet-in-the-middle attacks* on 3TDEA [227, 315] reduces the strength of 3TDEA to merely 112-bit equivalent security strength. In the case of 2TDEA, provided that the attacker can manage to gather approximately 2^{40} plaintext-cipher pairs, then 2TDEA would have a strength comparable to an 80-bit algorithm [257].

On the other hand and due to performance, functionality or compatibility reasons, algorithms of different strengths and key sizes are frequently combined in the same application. In general, the weakest algorithm and key size used for cryptographic protection determines the strength of the protection provided to the system. As an example, if SHA-512 is used with 1024-bit RSA, only 80-bit of security strength will be provided to data application. If the application requires 128 bits of security, then 3072-bit RSA key must be used. Alternatively, 256-bit ECC can be used to substitute RSA as a public key cryptographic engine.

Table 2.1 compares the security strengths of a set of algorithms divided into three categories: Symmetric block cipher algorithms, public key cryptosystems and hash functions. Notice, however, that novel or improved attacks and/or technologies may be developed in the future, leaving some of the algorithms included in Table 2.1 partially or completely broken. In particular, all hash functions listed in Table 2.1 have recently been subject of successful attacks, thus casting doubts on their security [368, 369, 103].

[9] The block size is also a factor that should be considered, since if a collision-attack is launched, collisions become probable after $2^{\frac{b}{2}}$ blocks have been encrypted with the same key for certain block ciphers' *modes of operation* [71, 70, 69].

Table 2.1. A Comparison of Security Strengths (Source: [258])

Private key Algorithm	**bit security**	**Expected Security lifetime**
Two-key triple DES	80	through 2010
Triple-key triple DES	112	through 2030
128-bit AES	128	beyond 2030
192-bit AES	192	beyond 2030
256-bit AES	256	beyond 2030
Public key Algorithm	**bit security**	**Expected Security lifetime**
DSA ($p = 1024, q = 160$)	80	through 2010
DSA ($p = 2048, q = 224$)	112	through 2030
DSA ($p = 3072, q = 256$)	128	beyond 2030
DSA ($p = 7680, q = 384$)	192	beyond 2030
DSA ($p = 15360, q = 512$)	256	beyond 2030
1024-bit RSA	80	through 2010
2048-bit RSA	112	through 2030
3072-bit RSA	128	beyond 2030
7680-bit RSA	192	beyond 2030
15360-bit RSA	256	beyond 2030
{160-223}-bit ECC	80	through 2010
{224-255}-bit ECC	112	through 2030
{256-383}-bit ECC	128	beyond 2030
{384-511}-bit ECC	192	beyond 2030
{512-}-bit ECC	256	beyond 2030
Hash functions	**bit security**	**Expected Security lifetime**
SHA-1	80	through 2010
SHA-224	112	through 2030
SHA-256,	128	beyond 2030
SHA-384	192	beyond 2030
SHA-512	256	beyond 2030

2.8 Potential Cryptographic Applications

During the last few years we have seen formidable advances in digital and mobile communication technologies such as cordless and cellular telephones, personal communication systems, Internet connection expansion, etc. The vast majority of digital information used in all these applications is stored and also processed within a computer system, and then transferred between computers via fiber optic, satellite systems, and/or Internet. In all those new scenarios, secure information transmission and storage has a paramount importance in the international information infrastructure, especially, for supporting electronic commerce and other security related services.

Under such a dynamic scenario, some of the most popular applications in the domain of information security include,

- *Secure e-mail*
- *World Wide Web*
- *Client-Server transactions*
- *Virtual Private Networks*
- *E-cash*
- *Electronic Financial transactions*
- *Grid Computing*

Many multinational firms now sell security products using cryptographic algorithms. Those products are in use by military or government organizations and they play a vital role in secure communications between individuals, small and large business groups.

Various international organizations have been working in developing standards for determining security and speed of products such as cellular phones, video conferencing equipment, secure telephone, etc. Examples include standards for video conferencing: H310, H323, H324 by ITU [154], for mobile communications: GSM by ETSI [87], for wireless LANs: 802.11a, 802.11b by IEEE LAN/MAN Committee [144], etc.

Numerous useful activities for increasing the security of cryptographic algorithms have happened in the few last years. The selection of the new Advance Encryption Standard (AES) 'Rijndael' and the inclusion of Elliptic curve cryptography (ECC) in international standards provide such examples.

Promising applications for cryptographic algorithms may be classified into two categories [250].

1. Processing of large amount of data at real time potentially in a high speed network. Examples include telephone conversation, telemetry data, video conferencing, streaming audio or encoded video transmissions and so forth.
2. Processing of very small amount of data at real time in a moderately high-speed network transmitted unpredictably. Examples include e-commerce or m-commerce transactions, credit card number transmission, order placement with signature, bank account information extraction, e-payments, and micro-browser-based (WAP-style) HTML page browsing and so forth.

A short list of the candidate applications corresponding to category 1 are presented in Table 2.2. Those applications belong to the "highly efficient" category of applications, thus requiring high data rates.

Table 2.2 presents both the downstream and upstream data transfer ranges on VDSL (Very high speed Digital Subscriber Line) [88, 252]. The downstream defines transmission of line terminal toward network terminal (from customer to network premise) and upstream in the reverse direction, that is, from network terminal to line terminal (from network to customer premise).

Table 2.2 can help to mark a line between high speed (highly efficient) and low speed (slow or relatively less speed) applications. The data rates for

Table 2.2. A Few Potential Cryptographic Applications

Application	Upstream	Downstream
Distance learning	384Kbps-1.5Mbps	384Kbps-1.5Mbps
Telecommuting	1.5Mbps-3.0Mbps	1.5Mbps-3Mbps
Multiple digital TV	6.0Mbps-24.0Mbps	64Kbps-640Kbps
Internet Access	400Kbps-1.4Mbps	128Kbps-640Kbps
Web hosting	400Kbps-1.5Mbps	400Kbps-1.5Mbps
Video conferencing	384Kbps-1.5Mbps	384Kbps-1.5Mbps
Video on demand	6.0Mbps-18Mbps	64Kbps-128Kbps
Interactive video	1.5Mbps-6.0Mbps	128Kbps-1.5Mbps
Telemedicine	6.0Mbps	384Kbps-1.5Mbps
High-definition TV	16Mbps	64Kbps

highly efficient applications ranges from 384Kbps to 24Mbits for upstream and 64Kbps to 3Mbps for the downstream traffic. From Table 2.2, the applications requiring a speed factor of less than 400Kbps can be grouped as low speed applications. Those applications require either stand-alone software implementations of cryptographic algorithms or the usage of software methods on embedded processors. High speed or highly efficient applications therefore reside in the range from 400Kbps onward.

Software methods on general-purpose processors cannot achieve such a high frequency gains for cryptographic algorithms. On the other hand, high speeds above 400Kbps can easily be achieved on both hardware platforms, the traditional (ASICs) and the reconfigurable hardware FPGA devices.

2.9 Fundamental Operations for Cryptographic Algorithms

Symmetric or secret key cryptographic algorithms are based on well-understood mathematical and cryptographic principles. The most common primitives encountered in various cryptographic algorithms are permutation, substitution, rotation, bit-wise XOR, circular shift, etc. This is one of the reasons for their fast encryption speed. On the other hand, asymmetric or public key cryptographic algorithms are based on mathematical problems difficult to solve. The most common primitives in various such types of algorithms include modular addition/subtraction, modular multiplication, variable length rotations, etc. Those primitives give algorithmic strength but they are hard to implement: occupy more space and consume more time.

Therefore those algorithms are not used for encrypting large data files, but rather, they are applied to other important cryptographic applications like key exchange, signature, verification, etc.

A detail survey conducted in [44], identifies the basic operations involved in several cryptographic algorithms. That survey has been slightly updated as shown in Table 2.3.

Table 2.3. Primitives of Cryptographic Algorithms (Symmetric Ciphers)

Modular addition or subtraction	Blowfish, CAST, FEAL,GOST, IDEA, WAKE RC5, RC6, TEA, SAFER K-64, Twofish, RC4 SEAL, TWOPRIME
Bitwise XOR	Blowfish, CAST, DEAL, TWOPRIME, FEAL, A5 IDEA, GOST, RC4, RC5, SAFER, SEAL, Twofish DES, WAKE, LOKI97, LOKI91, Rijndael, MISTY TEA, MMB, RC6, K-64
Bitwise AND/OR	MISTY
Variable-length rotations	CAST, Madryga, RC5, RC6
Fixed-length rotations	DEAL, DES, CAST, FEAL, GOST, Serpent, RC6 Twofish
Modular multiplication	CAST, IDEA, RC6, MMB, Rijndael,
Substitution	Blowfish, DEAL, DES, LOKI91, LOKI97, Twofish Rijndael
Permutation	DEAL, DES, ICE, LOKI91, LOKI97
Non-circular shifts	Serpent, TEA

From Table 2.3, it is clear that most cryptographic algorithms include bitwise operations such as XOR, AND/OR, etc. Those operations can be nicely implemented on hardware platforms. Long word length is another peculiarity of cryptographic algorithms, which is recommended by various international standards in order to attain sufficient security against brute force attacks.

The long key/word length of cryptographic algorithms is an obstacle for parallel dataflow on 8, 16, 32-bit general-purpose processors resulting on high time delays for the execution of crypto algorithms. This is not the case for hardware implementations. For example, in FPGAs, more than 1000 input/output pins are available for their use as either input or output buffers allowing high parallelism of data [392, 394].

In order to *confuse* the relationship between input and output, cryptographic algorithms perform a number of iterations on the same input data block for one encryption. DES performs 16 iterations or rounds and AES support 10, 12, and 14 rounds depending on the word length. In software, all iterations are performed sequentially while in hardware, all rounds can be implemented in parallel, thus ensuing significant improvements in timings.

2.10 Design Alternatives for Implementing Cryptographic Algorithms

The implementation approaches for cryptographic algorithms are based on the question: what needs to be secured?

High-speed network where large amount of data traffic must be processed in unpredictable and in real time are not supposed to be a good candidate for software implementations as data is coming at significant high speeds and must be treated in real time. Examples of such situation include telephone conversation, video conferencing, and so forth.

Hardware solutions on VLSI can accommodate high data rates but they take long development cycle for the application. Any change or modification in the design is a difficult or even impossible task.

A hardware solution that overcomes the difficulties of VLSI designs, while still allowing high dataflow, is reconfigurable hardware platforms. Indeed, Reconfigurable hardware devices such as FPGAs (Field Programmable Gate Arrays) provide fast solutions in short time with a high degree of flexibility.

Table 2.4 presents a quick comparison of reconfigurable logic against software and VLSI based solutions.

Table 2.4. Comparison between Software, VLSI, and FPGA Platforms

	Software	VLSI	FPGAs
Size	small (depends)	big	small
Cost	low	high cost	low cost
Speed	low	Very high	high
Memory	fine	fine	fine
Flexibility	highly flexible	no flexibility	highly flexible
Time-to-market	short	very high	short
Power consumption	depends	low	high
Testing/Verification	easy	difficult	easy
Run-time configuration	none	none	yes

Software implementations are low cost, easy to debug, take short time cycle but are slow. VLSI implementations are very fast but their application development cycle is too large and also they are not flexible. Reconfigurable devices are fast, highly flexible, easy to debug and take small developing cycle offering cost effective solutions.

In summary, using reconfigurable hardware for cryptographic algorithms is beneficial in several ways:

- Most cryptographic algorithms, especially symmetric ciphers, contain bit-wise logic operations whose implementation fits very well on the FPGA CLB structure.

- In Section 2.9, it was mentioned the iterative nature of most cryptographic algorithms. An iterative looping design (IL) implements only one round. Hence, n iterations of the algorithm are carried out by feeding back previous round results. For a high speed network, instead of implementing one round, n rounds of the algorithm can be replicated and registers are provided between the rounds to control the flow of data. Reconfigurable FPGA logic results useful for both design strategies due to its high speed and high-density features.
- Substitution is the fundamental operation in most block ciphers like DES or Rijndael which implies the usage of lot of memory resources. The usage of pipeline design strategies, tend to provoke significant memory requirements. Fortunately modern FPGA families like Xilinx Virtex series device are equipped with more than 280 built-in memory blocks 4K each, called *BlockRAMs* (BRAM).
- At the same time, in several contexts, designers may use reconfigurable FPGA logic to implement in the same hardware both the public key algorithm for the generation and secure exchange of key and the private key algorithm traditionally used in the bulk encryption of the underlying traffic.
- The usage of different cryptographic algorithms for various applications faces several compatibility issues. A dynamic configuration for any cryptographic algorithm on FPGA might be a good compromise solution to this problem.
- FPGA devices are ideal for debugging and fast prototyping, especially if the synthesized hardware description can be mapped by the design team from FPGA domain to ASIC.
- The flexibility for integration into larger platform together with straightforward architecture modifications are significant pluses for FPGA platform implementations.

2.11 Conclusions

In this Chapter we gave a short introduction to the algorithms and security services corresponding to layers 2-4 of Fig. 2.1. This way, basic concepts of cryptography along with a description of the main building blocks necessary for constructing security applications was given. We described the basic operation of symmetric block ciphers, hash functions, three major public key cryptosystems and the celebrated Diffie-Hellman key-exchange protocol. We also gave some comments on the security provided by the main cryptographic schemes and their equivalent security strength. Furthermore, alternatives for the implementation of cryptographic algorithms on various software and hardware platforms were also analyzed and discussed.

As a conclusion, we believe that Reconfigurable logic offers numerous useful advantages, however its usage in inexpensive consumer-oriented devices

such as electronic gadgets, wireless PDAs and handsets seems to be impossible at present time.

On the contrary, FPGA devices can be contemplated on embedded systems, large wireless equipments, electronic transmitters and receivers, repeaters, spectrum scanning devices, and intelligent equipment.