



A short introduction to modern cryptography

- explanatory supplement to the cv act *library/es* -



Table of contents

1	Introduction	3
1.1	Scope of this documentation	3
1.2	Organization of chapters	3
1.3	General principles	3
2	Symmetric cryptography	5
2.1	Symmetric encryption	5
2.2	Hash functions	6
2.3	Message authentication codes (MACs)	6
3	Asymmetric Cryptography	8
3.1	The foundation of public key cryptography	8
3.2	Key exchange	9
3.3	Asymmetric encryption	9
3.4	Digital signatures.....	10
4	Additional mechanisms	12
4.1	Random number generation.....	12
4.2	Padding.....	12
5	Elliptic Curve Cryptography	13
5.1	Domain parameters.....	13
5.2	Keys.....	13
5.3	Signatures.....	14
6	Cryptographic security and key lengths	15
6.1	General aspects	15
6.2	The model of Lenstra and Verheul	15
6.3	Practical threats in real-world scenarios	16
	Bibliography	18

Provided by: cv cryptovision GmbH
Munscheidstr. 14
45886 Gelsenkirchen
Doc-Ref: Modern_Cryptography.doc
Version: 1.0
Date: 04/2005

1 Introduction

This short introductory document on modern cryptography is meant as a guideline for embedded system developers that want to secure their applications using cryptovision's **cv act library/es** [4].

While by far not a complete survey on cryptography, this introduction provides basic background information on various cryptographic mechanisms and how to employ them in a secure way.

1.1 Scope of this documentation

This guideline is meant for engineers and software developers working in the field of embedded systems. It provides basic information about cryptography and its application (especially for computationally restricted environments). As such, this text assumes a basic knowledge about data processing and the appraisal of application specific information worth protecting.

This document complements the technical description of the **cv act library/es** [4] and shall be used as a general reference. The dedicated information about the functions provided by this library, including prototypes, data formats, code sizes and execution times can be found in this technical description.

1.2 Organization of chapters

This document is organized as follows:

- Section 1.3 explains the general principles of modern cryptography.
- Chapter 2 provides background information about the classic domain of symmetric cryptography, while chapter 3 contains an insight into modern asymmetric (public key) mechanisms and illustrates the concept of key exchange and digital signatures.
- Chapter 4 surveys fundamental additional mechanisms, that is random number generation and data formatting by (random) padding.
- Chapter 5 provides an introduction into advanced public key mechanisms based on *Elliptic Curves*, that allow for faster execution and shorter parameters compared to classic asymmetric cryptography while preserving the same level of security.
- Chapter 6 elaborates on cryptographic security and key lengths as well as on practical threats in real-world scenarios.

1.3 General principles

Information is the raw material of the 21st century. The exchange and gathering of information has become a central component of private and economic activities. Information is filed in documents and is transmitted by communication. If confidential information gets into the wrong hands, immense damage could be done.

Today, the protection against misuse, manipulation and eavesdropping is already considered the basic challenge of this era. New cryptographic mechanisms, which can be used in a flexible way, work efficiently and provide a high level of security, are the key to this problem.

Cryptography provides means to guarantee the following critical issues of information and communication:

- **Confidentiality:** a message should be protected from being read by non-legitimate persons.
- **Authentication:** the receiver should be able to identify the origin of a message.
- **Integrity:** the receiver should be able to examine whether a message was changed during transmission.
- **Liability:** a sender should not be able to deny having sent a message.

Today, one distinguishes between two fundamentally different approaches to cryptography: **symmetric** and **asymmetric** (or public key) mechanisms.

These mechanisms will be explained in the following two chapters; for illustration, the communication setting between two imaginary persons – traditionally called Alice and Bob – will be used.

2 Symmetric cryptography

The defining property of this class of cryptographic mechanisms is that the sender and receiver of a secured communication use the same key for its protection. Thus for each communication link the communicating partners have to share a common secret key. The main drawback of this approach is the number of necessary keys (i.e. one per communication partner) that have to be stored by each participant and the problem to securely agree on or exchange an appropriate key to set up a secure communication link.

Modern symmetric mechanisms are typically based on rather simple operations, like bit shifts and XORs and are therefore quite efficient.

The most common application of this class of cryptographic mechanisms is the *symmetric encryption*.

2.1 Symmetric encryption

In symmetric encryption a *plaintext message* is *encrypted* by the sender using a secret key. The resulting *cipher text* can then be *decrypted* by the receiver with the same (or a closely related) key while non-legitimate persons can not reconstruct the original message from the cipher text without this key. Symmetric encryption hence provides confidentiality of the transmitted information.

Stream and block ciphers

Larger messages usually have to be split up before encryption, as this procedure requires an input of a fixed length. In the literature, symmetric algorithms are divided into two categories:

The first process the plaintext character-wise and are called stream algorithms or *stream ciphers*. In this context, a character could be e.g. a bit or a byte. The others process the plaintext in bit-groups which are called *blocks*. They are called block algorithms or *block ciphers*. Stream ciphers are usually faster in hardware implementations than block-oriented procedures, because the bit- or byte-wise processing is better adopted to a hardware solution. In software-based systems block ciphers are usually used. However, it is also useful for block ciphers to make the ciphering dependent on the processing of previous blocks. This provides that identical plaintext blocks are copied to different cipher texts and, thus, make work more difficult for a potential attacker.

This chain of successive blocks is called a *cryptographic mode*. The different versions include *electronic codebook* (ECB), *cipher block chaining* (CBC), *cipher feedback* (CFB) or *output feedback* (OFB). In the ECB mode a codebook is created including plaintexts and corresponding cipher texts. It denotes the mode without connection. In the CBC mode, on the other hand, a plaintext block is XOR connected to the previous block before encryption.

DES

DES is the classic among cryptographic procedures and represents the creation of modern cryptography. It was developed at IBM as a result of a tender of the *National Bureau of Standards* (NBS, today NIST) for a uniform encryption standard and was presented in 1976. This presentation was a small sensation at that time, as DES was the first standardized and, above all, publicly made known encryption procedure for really high security requirements. Every 5 years, NIST certified the DES algorithm. The last certification was carried out in 1999, yet under the condition that the DES version Triple-DES is used, as DES no longer meets today's security requirements.

AES

The *National Institute of Standards and Technology* (NIST) established a DES successor as the new standard for symmetric encryption procedures: AES, Advanced Encryption Standard. The main objective is the establishment of a *Federal Information Processing Standard* (FIPS). On 12 September 1997, a formal tender for algorithms has been published. Certain conditions for the algorithm included, among others:

- a symmetric cryptographic procedure
- a block cipher
- a block length of 128 bits
- a key size of 128, 192 and 256 bits

The AES finalists of the third and final round were:

- MARS
- RC6
- Rijndael
- Serpent
- Twofish

From this finalists Rijndael was selected on 2 October 2000 as AES.

The AES encryption algorithm (cf. [7]) is included in the **cv act** *library/es*.

2.2 Hash functions

Though hash functions do normally not involve a cryptographic key – and are therefore strictly speaking not symmetric mechanisms – they are realized using the same construction principles.

A hash function compresses data in a specific way which can hardly be inverted in practice. The exact definition requires the use of the term *one-way function*: This is the name of a function f which can be calculated quickly but for which it is practically not possible to determine an argument x with $f(x) = y$ for a given functional value y . Consequently, these are functions which can quickly be calculated but of which the inversion is significantly more difficult.

A (cryptographic) *hash function* is a one-way function which provides for an input of any length a usually shorter output of a fixed length (the *hash value*). This compression feature of hash functions is useful for many applications. One example is the calculation of electronic signatures where instead of the complete input data only its hash value needs to be subjected to the signature process (cf. section 3.4).

The **cv act** *library/es* provides the *Secure Hash Algorithm (SHA-1)* as defined in [5].

2.3 Message authentication codes (MACs)

For an electronic authentication of data, the message M is expanded by the special redundant information which is calculated from M using a cryptographic procedure, and is stored or transmitted together with the message. In order to ensure that the attacker is unable to modify the attached redundancy directly, it has to be protected appropriately.

If the transmission of the redundant information is carried out using a secret key, this is referred to as a *Message Authentication Code (MAC)*. Therefore, a MAC is also referred to as a hash function with an additional secret key. The theory behind MACs does not differ from the theory behind hash functions, besides that in this case the hash value can only be verified with the corresponding key.

A MAC is used to guarantee integrity and authenticity of a message M . To fulfill these aims the following characteristics are required:

- $\text{MAC}(K, M)$ can “easily” be calculated with the given message M and the key K .
- It is practically not possible to determine the key K from the message M and the corresponding authentication code $\text{MAC}(K, M)$.
- It is practically not possible with the given message M and given $\text{MAC}(K, M')$ to find a message M' different from M which provides the same authentication code as M , i.e. with $\text{MAC}(K, M) = \text{MAC}(K, M')$.

In every case the proof of the integrity of a message is based on the secrecy or integrity of the cryptographic key. The most simple case is to encrypt the hash value with a symmetric algorithm. A potential security problem is that the receiver has to know the key. With this key, however, he can create other messages with the same hash value.

A MAC can also be created from a hash function or a symmetric block cipher (CBC-MAC). The **crypto** **act library/es** provides a hash-MAC based on the Secure Hash Standard SHA-1.

3 Asymmetric Cryptography

Today, cryptography has moved from an originally military domain to a broad area of application and is – often invisibly – involved whenever (electronic) communication or information has to be secured.

With the rapidly increasing number of participants and new applications beyond closed user groups, the above mentioned drawbacks of symmetric cryptography concerning key handling and key storage became even more critical.

Fortunately, these problems were overcome with the invention of *public key cryptography* in the 1970s. While in the symmetric setting, the same (secret) key is used for encryption and decryption (as well as MAC generation and verification), these functions are separated from each other in the public key scenario. Here, the secret key is split into a private and a public key part. The private key has still to be kept secret, whereas the public key is published in an appropriate register. The receiver's public key is used for encryption and verification of digital signatures (see section 3.4) while the private key is used for decryption and signature generation.

To illustrate the – somewhat paradoxical – concept of asymmetric encryption, one may think of putting the written plaintext in a footlocker and securing it with a padlock provided by the intended receiver. While everyone that has such a padlock (corresponding to the receiver's public key) can lock the footlocker (i.e. “encrypt” the message), only the receiver can open it using the (private) key (i.e. “decrypt” the message).

The most popular asymmetric cryptographic system is the RSA system, named after its inventors Rivest, Shamir and Adleman. While RSA is widely used in general IT settings, it is rather cumbersome and difficult to be implemented for restricted environments due to its huge parameters (cf. next subsection). Alternative systems, of which *Elliptic Curve Cryptography (ECC)* is certainly the most common one, better suit the demands and restrictions of embedded systems.

The **cv act library/es** provides elliptic curve key generation, Diffie-Hellman key exchange (ECDH) according to ANSI X9.63 (cf. [2]) and digital signatures based on the Elliptic Curve Digital Signature Algorithm (ECDSA) according to ANSI X9.62 (cf. [1]).

3.1 The foundation of public key cryptography

The fundamental building block of asymmetric cryptography is known to be a *trapdoor function*. Such a function is easy to compute but computationally impossible (or at least very difficult) to invert unless one has a special *trapdoor information* (the private key).

Most practical realizations of trapdoor functions are based on intractable mathematical (more precisely number theoretic) problems. In case of RSA this is the problem of decomposing a large composite number into its prime factors called the *integer factorization problem (IF)*. ECC and DSA, on the other hand, are based on two different variants of the *discrete logarithm problem (DLP)*, which is the inversion of the (easily computable) exponentiation in the respective mathematical structure.

To provide a sufficient gap between the costs to perform the “easy direction”, e.g. multiplication or exponentiation and the costs needed to solve the “inverse” problem, one has to set up huge instances of these problems using long integers (with currently up to 500 decimal digits). Since the best known algorithms for computing discrete logarithms in the ECC setting are on principle less efficient than those for DSA and the integer factorization problem, ECC parameters can be chosen significantly shorter for the same level of security.

3.2 Key exchange

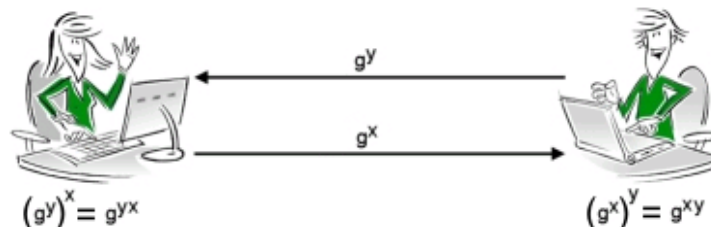
The following protocol – introduced in 1976 by W. Diffie and M. Hellman – is the historic starting point of public key cryptography and solves the problem of exchanging a common secret key over an insecure channel.

It is based on the above mentioned discrete logarithm problem in a mathematical structure called the *multiplicative group of a finite (prime) field* (typically denoted Z_p^*). This is the set of integers between 1 and $p - 1$, for a prime p , together with an operation called *modular multiplication*. This operation maps a pair of these numbers x, y on the *modular product* $x * y \pmod{p}$, which is – by repeatedly subtracting p from the “ordinary product” – again a number of this set. So as an example, $5 * 6 \pmod{13} = (30 - 2 * 13) = 4$. This operation has all the properties we know from the multiplication on the (non-zero) real numbers. By n -fold modular multiplication of an element x , we get the *modular exponentiation by n* , $x^n \pmod{p} := x * x * \dots * x \pmod{p}$. Now in each such a group there are elements g , whose powers $g^1, g^2, \dots, g^{(p-1)}$ run through all elements of this group. These special elements “generate” the whole group and are hence called *generators*.

The abstract problem of the discrete logarithm mentioned before can now be expressed explicitly: Given a generator g and an (arbitrary) element y of such a group, find the distinct element x , such that $g^x = y \pmod{p}$. While modular exponentiation can easily and efficiently be computed, this problem remains difficult and can not be solved efficiently if p is large enough.

With that said, the key exchange protocol can be stated as follows:

1. Alice and Bob agree on a suitable group Z_p^* and a generator g .
2. Each of them chooses a secret number x, y from Z_p^* , respectively.
3. Alice computes $g^x \pmod{p}$ and Bob $g^y \pmod{p}$ and they exchange their results.
4. Both can compute the same common secret $(g^y)^x = g^{yx} = g^{xy} = (g^x)^y$ using their private information; this secret can be used as a shared key for symmetric encryption.



An eavesdropper trying to deduce the common secret from g^y and g^x would have to compute either $(g^y)^x$ or $(g^x)^y$ without knowing x or y , respectively. It is generally thought that it is necessary to compute the corresponding discrete logarithm to do so.

3.3 Asymmetric encryption

Sending an encrypted message in the public key setting is carried out as follows:

- Alice obtains Bob's public key in a trustworthy way.
- Alice enciphers her message using Bob's public key and sends the message to Bob.
- Bob decipheres the message using his private key.

In contrast to symmetric mechanisms, one of the major weak points for secured communication is cleared out, i.e. the key exchange via an insecure channel. The potential receiver, Bob in this case, simply publishes the partial key required for encryption and keeps the other, private key in his pos-

session. This provides that anyone can encrypt data, but only the legitimate receiver is able to decrypt it.

Furthermore, the number of required keys decreases in asymmetric settings: For n communication partners the management of a database readable for everyone with n keys is sufficient (in contrast to roughly n^2 keys in a symmetric setting). This, however, reveals a basic weak point of these Public Key mechanisms: Alice must obtain Bob's public key. It is a worthwhile target for an attacker to trap and change information transmitted during the database inquiry. There exist various techniques to guarantee authenticity of public keys, typically relying on a trusted third party called the *certification authority (CA)*. This authority issues certificates that affirm the connection between someone's identity and the corresponding public key. These techniques are subsumed under the term *Public Key Infrastructure (PKI)*.

In practice, however, public key algorithms are no replacement for symmetric algorithms as they are by far too slow. This is why they are used e.g. for the ciphering of keys for symmetric procedures. These so-called *hybrid ciphers* represent cryptographic mechanisms where a public key is used for protecting and distributing session keys (used only for one or few communication sessions), which are then employed with a symmetric cipher for fast encryption.

The protocol then works as follows:

- Alice obtains Bob's public key in a trustworthy way.
- Alice generates a random session key, encrypts it using Bob's public key and sends it to Bob.
- Bob decipheres Alice's data using his private key to obtain the session key.
- They both encrypt/decrypt their communication using a symmetric cipher and the same session key.

3.4 Digital signatures

The *digital signature* of an electronic document is the equivalent to a handwritten signature on a document in paper. Considering the increasing importance of electronic media, it is required to have such an equivalent to a traditional signature available. The problems involved comprise the following questions:

- Has the document been changed after completion?
- Does the document come from the given sender?

In addition, the legal stipulations also have to be considered. In Germany, the Signature Act and an supplementary Signature Decree were passed in 1997, which provide, among others, the legal framework conditions for false-proof digital signatures and reserve the rights of participants in electronic legal transactions.

Also on EU echelon there have been several approaches in the recent past for regulations concerning digital signatures. On 19 January 2000 the regulation 1999/93/EG was published and became effective therewith. The EU member states have now 18 months (until 19 July 2001) to translate this regulation into national law.

There are algorithms for signing documents using Public Key cryptography. In an asymmetric encryption procedure one usually uses the public key for encryption and the private key for decryption. With digital signatures the use of the keys is reversed:

- Alice enciphers data using her private key.
- Alice sends the signed document to Bob.
- Bob decipheres the document using Alice's public key and verifies the authenticity of the signature at the same time.

Signing large documents using an asymmetric mechanism this way takes some time, though. Hence, in practice, one typically signs a short hash value of the document instead of the document itself (similar to a hybrid mechanism).

The protocol for generating digital signatures now looks as follows:

- Alice ciphers the hash value of her document using her private key signing the document.
- Alice sends the document and the signed hash value to Bob.
- Bob calculates the hash value of the document sent by Alice using the same hash function. With Alice's public key he deciphers the signed hash value. If it corresponds to the hash value he created, he can trust the authenticity of the document.

If you take a look at the questions listed above you can see that the problem of falsification has already been solved by the application of hash functions which make these changes detectable. The other problem has been solved by the application of an asymmetric cryptographic system, as the identity of a person is ensured by his or her public key, if this has been confirmed by a certification authority within a public key infrastructure.

In contrast to a (symmetric) message authentication code, where the authorship of the document in question can only be retraced to a group of communication partners sharing the same secret key, it can be narrowed down to the single person with the corresponding private key. For this reason, the digital signature even allows the undeniable proof of authorship to a third party, a property called *non-repudiation*.

4 Additional mechanisms

Apart from the above mentioned ingredients, there are some additional mechanisms used in cryptography.

4.1 Random number generation

Random numbers play a central role in cryptography, as they are used in various primitives, e.g. key generation, digital signatures and challenge & response protocols. The security of an entire cryptographic system may depend on the quality and unpredictability of the random numbers used. True random sequences can be derived from chaotic physical processes like radioactive decay or the noise of a free-running oscillator. These sources tend to be expensive and rather difficult to be integrated into standard computer environments, however.

Besides these true random sources, one often uses a *pseudo random number generator (PRNG)*. This is a deterministic process that produces a seemingly random output of arbitrary length based on a short (truly) random input, called the *seed*. For cryptographic purposes such a pseudo random number generator should have the following characteristics:

- It seemingly creates random outputs, i.e. it must pass all statistic tests known.
- The pseudo random sequence is not predictable. It must be impossible to calculate which random bit follows next, even if the algorithm or the hardware creating the sequence as well as all previous bits are known.
- The generator cannot be reproduced in a reliable way. If a process is carried out twice, one is to receive two sequences having no similarity.

As the generation process is deterministic, the uncertainty (often called *entropy*) of the output sequence is determined by the seed only and this seed must hence be carefully chosen.

The **cv act library/es** provides a pseudo random number generator according to FIPS 186-2 [6].

4.2 Padding

Some cryptographic mechanisms like block ciphers or digital signatures expect inputs of a fixed length. If the actual input is too short, it must therefore be extended appropriately; this process is called *padding*. While some mechanisms require a deterministic padding, that guarantees that the reconstructed message is identical to the original input, other padding techniques involve random numbers, e.g. to ensure that reapplying an encryption on the same plain text leads to different cipher texts.

The cryptographic protocols provided by the **cv act library/es** involve various standard padding mechanisms, that are explained in the respective technical description, where necessary.

5 Elliptic Curve Cryptography

This section provides a brief overview of the main components of a cryptographic system based on elliptic curves.

5.1 Domain parameters

To set the stage for an ECC system, one has to fix a set of domain parameters. This public information specifies the underlying mathematical structure (called a *finite field*), the chosen elliptic curve over this field and the group (and its generator G usually called *base point*) used for cryptographic computations. A set of domain parameters consist of the following entries:

p	prime number specifying the underlying field $\text{GF}(p)$
a	first coefficient of the Weierstraß equation defining E
b	second coefficient of the Weierstraß equation
G	base point, $G = (G_x, G_y)$
n	order of G
h	cofactor ($\#E = nh$)

Note: This library requires domain parameters with cofactor $h = 1$, i.e. $n = \#E$.

The order n of this base point (or its bit length) – which is of the dimension of the order of the elliptic curve – defines the security of the ECC system. As a rule of thumb, the security provided by an “160-bit” elliptic curve is comparable to the security of RSA with a 1024-bit modulus; for further notes on cryptographic security, please refer to chapter 6.

Appropriate domain parameter sets for common bit lengths are published by various standardization bodies, like NIST, ANSI, ISO and SECG; cryptovision provides some of these parameter sets along with this library. In addition to those sources, it is possible to create individual domain parameters (see [9] for details).

Even though one set of domain parameters is typically used by all participants of an ECC system, it is possible (and supported by this library) to use several distinct sets of parameters. This is helpful in the seldom case, that a (embedded) system is used for several applications with different domain parameters or that a public key – based on a different set of parameters – has to be used for verification.

Since the appropriate choice and the correctness of the domain parameters are of central importance for the system’s security (especially if they are used to create a private key), it is advised to use parameters provided by a trustable source or to validate these parameters before use. Since this process incorporates heavy computation that is beyond the capabilities of a typical embedded system, it has to be provided from outside. For validation of domain parameters, one can use cryptovision’s general purpose **cv act library**; for further information on this topic, please refer to [9].

As already indicated before, the domain parameters are typically public and have to be accessible from outside the smartcard; they have to be protected against modification by an appropriate definition of access rights, though. In the seldom case that the domain parameters shall be kept secret, a similar access definition has to be used to prevent the respective file from being read out.

5.2 Keys

An ECC key pair consists of a private and a public key part, which are generated together and are linked with the domain parameters. The private key d is a randomly chosen integer, $1 \leq d \leq n-1$.

The public key Q consists of the scalar product of this private key d and the base point G , $Q := d * G$.

The private key is used to generate signatures and must therefore be kept secret. It is usually generated within the embedded system to be stored in a secured fashion and never leaves this protected environment.

To verify a signature, the public key of the respective signer is needed. A correct signature provides evidence, that the signature was generated with the corresponding private key. Therefore it is important, that nobody can manipulate the public key.

There are various techniques used to guarantee the authenticity and consistency of public keys, involving trusted third parties (trust centers) or decentralized webs of trust, for example; these topics are however beyond the scope of this manual; see [13] for details.

5.3 Signatures

An ECDSA signature is a pair (r, s) of long integers satisfying a pair of certain equations.

Due to their probabilistic nature (introduced by a randomly chosen parameter involved in the signature generation process), signatures with the same private key and the same message differ from each other.

6 Cryptographic security and key lengths

This section provides some background information on the security assessment of cryptographic systems.

6.1 General aspects

Usually the cryptographic security of a given encryption algorithm is a function of the key length. The cryptographic security corresponds to the calculation effort that is needed to compute the key out of a pair of given plain text and cipher text, for example.

It should be underlined that in real systems security problems are often not caused by the failure of the encryption algorithm, but by weaknesses in the protocol or the implementation. For example, today a 512 bit RSA encryption is generally assumed to be insecure. Nonetheless, breaking a 512 bit RSA key (which is equivalent to the factoring of a 512 bit number) is still not a trivial task and is far beyond the scope of a single hacker (although a larger group, e.g. organized via internet, could attack it successfully). On the other hand, the chance that a hacker finds the key with a conventional attack (e.g., Trojan horse) on a usual PC platform is comparably high.

Basically, there are three different types of cryptographic primitives that have to be evaluated: symmetric encryption algorithms, asymmetric encryption algorithms, and hash functions. The best-known symmetric algorithm is DES, introduced in 1977, with a key size of 56 bit. Although the first successful attack on DES was publicly demonstrated in 1997 (a parallel search, taking 4 months on 3500 computers), concepts for special computing platforms to retrieve the DES key were discussed in the academic field long before, and it is generally assumed that large institutions (e.g., the National Security Agency) had the computing power to retrieve DES keys at least in the early 80's.

In contrast to symmetric algorithms, asymmetric encryption algorithms (also known as public key algorithms) are based on special mathematical problems. Examples are RSA or ECC (elliptic curve cryptography) algorithms. Due to different underlying mathematics, the key sizes for the same level of cryptographic security can differ a lot. As an example, RSA encryption with 1024 bit key length leads to a security that is comparable to an ECC encryption with a key length of approximately 135-160 bit.

The third group of important cryptographic primitives, hash functions, generate a cryptographic checksum (hash value). The equivalent of the key length in encryption is the length of the resulting hash value. Typical values are 128 or 160 bit. A successful attack on a hash function would result in a data string that produces a given checksum. Another, simpler class of attack would result in the finding of two data strings that produce the same checksum (which is simpler because of the birthday paradox). In the latter case, a level of security roughly comparable to a symmetric algorithm of given key length can be achieved by a hash function that produces hash values of twice the key length.

6.2 The model of Lenstra and Verheul

The standard model for the evaluation of cryptographic key sizes was developed by Lenstra and Verheul [12]. This model is, for example, used by the German Information Security Agency (GISA, German: *BSI*) as a basis for the annual list of recommended cryptographic algorithms for the German signature law. Based on a review of the known attacks on the different encryption methods, Lenstra and Verheul applied, among others, the following two basic assumptions:

- constant increase of the available computation power according to Moore's law (doubling every 18 months)
- constant refinement of the available mathematical methods in the case of asymmetric algorithms

The model needs a starting point for calculations. The standard table of Lenstra and Verheul uses the DES algorithm for comparison. The original table presents the necessary bit lengths to achieve a security relative to that of a DES encryption in 1982. In other words, the level of security is relatively high, so high that no private or public institution should have the possibility to break the system. Of course, this is a rather high limit, e.g. in business communication DES encryption was used throughout the nineties without any known problems.

The following table lists some results of Lenstra and Verheul. The key sizes in every line are comparable because the same calculation effort (e.g., measured in MIPS-years) is necessary to break the encryption. It should be noted that the work of Lenstra and Verheul contains many parameters to adapt the model to predicted technical developments; our numbers are based on the authors own assumptions. The original paper contains two different estimations for ECC key length, based on moderate and drastic scientific progress concerning methods for solving the underlying mathematical problem, respectively; here, the mean of these two values is provided.

Year	Symmetric key size	RSA key size	ECC key size	MIPS-years needed for attack
2000	70	952	132	$7.13 * 10^9$
2002	72	1028	137	$2.06 * 10^{10}$
2004	73	1108	141	$5.98 * 10^{10}$
2006	75	1191	145	$1.73 * 10^{11}$
2008	76	1279	149	$5.01 * 10^{11}$
2010	78	1369	153	$1.45 * 10^{12}$
2012	80	1464	157	$4.19 * 10^{12}$
2014	81	1562	162	$1.21 * 10^{13}$
2016	83	1664	166	$3.51 * 10^{13}$
2018	84	1771	170	$1.02 * 10^{14}$
2020	86	1881	175	$2.94 * 10^{14}$
2022	87	1995	179	$8.52 * 10^{14}$
2024	89	2113	184	$2.47 * 10^{15}$
2026	90	2236	188	$7.14 * 10^{15}$
2028	92	2362	192	$2.07 * 10^{16}$
2030	93	2493	196	$5.98 * 10^{16}$

Table 1: Comparison of key lengths and their security

It should be pointed out that a forecast of future developments in computing power and algorithmic developments can only be a vague approximation. Therefore, especially the numbers for the far future should be used with caution.

6.3 Practical threats in real-world scenarios

When designing a system that may be attacked by a third party, the adequate choice of the cryptographic mechanisms and parameters can not guarantee practical security. Besides the cryptographic algorithms itself, attackers can focus on the storage of cryptographic keys; they may find weaknesses in the chip design of an embedded system or in the operating system, which are much easier to attack.

An example: In some early applications of chipcards, attacks were possible by applying unusual temperatures or voltages, or by resetting the chip during the calculation process. Today's high-security smartcards have special operating systems and countermeasures against attacks like this, but this will not be the case for a standard embedded system. Thus, the system designer has to

keep the possible threats in mind, and the win and the costs that are necessary for an attacker. Will a specific manipulation of a system result in one single manipulated hardware unit, or can the result be exploited in a way that threatens the security of all systems in the field? All these potential risks have to be analyzed and should influence the design. A secure embedded system has to be based on sound cryptographic mechanisms as well as a secure overall system design.

One additional aspect designers of embedded systems for a high security level should keep in mind are some new kinds of attacks developed in recent years. These so-called side-channel attacks make use of information which is unintentionally disclosed during cryptographic calculations. This may be the calculation time, which can for example be influenced by a Hamming weight of a cryptographic key, but also the power consumption during the calculation, or the electromagnetic emanation of the chip. The last two examples can show enough information to reconstruct the basic operations of the chip during cryptographic calculations, and they may also give direct information about the cryptographic keys.

In the field of high-security smartcard applications, a large set of countermeasures against such attacks are implemented. Because every processor has its own characteristics, the countermeasures are usually specifically implemented in assembly language. The **cv act library/es** also contains a set of general countermeasures, which makes it superior compared to standard cryptographic libraries. Nonetheless, if it is planned to use the library high-security applications with a high attack risk (e.g., in applications like large-scale pay TV), we recommend to contact [cryptovision](#) and to discuss the additional integration of processor-specific countermeasures.

Bibliography

- [1] ANSI X9.62: *Public Key Cryptography for the Financial Service Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)*, American National Standards Institute (1999)
- [2] ANSI X9.63: *Public Key Cryptography for the Financial Service Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography*, American National Standards Institute (1999)
- [3] Blake, I.; Seroussi, G.; Smart, N.: *“Elliptic Curves in Cryptography”*, Cambridge University Press (1999)
- [4] cv cryptovision GmbH: cv act library/es, Cryptographic Library for Embedded Systems - Technical Information, 2004-2005.
- [5] FIPS 180-1: *Secure Hash Standard*, Federal Information Processing Standards Publication 180-1 (1995)
- [6] FIPS PUB 186-2: *Digital Signature Standard (DSS)*, Federal Information Processing Standards Publication 186-2 (2000)
- [7] FIPS PUB 197: *Advanced Encryption Standard*, Federal Information Processing Standards Publication 197 (2001)
- [8] FIPS PUB 198: *The Keyed-Hash Message Authentication Code (HMAC)*, Federal Information Processing Standards Publication 198 (2002)
- [9] IEEE P1363: *Standard Specification for Public Key Cryptography*, Draft (1999), available online at <http://grouper.ieee.org/groups/1363/index.html>
- [10] ITU-T X.680: *Information technology – Abstract Syntax Notation (ASN.1) : Specification of basic notation*, International Telecommunication Union (2002)
- [11] ITU-T X.690: *Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*, International Telecommunication Union (2002)
- [12] Lenstra, A.; Verheul, E.: *Selecting cryptographic Key Sizes*, Journal of Cryptology, Springer Verlag (2002), available online at <http://www.cryptosavvy.com>
- [13] Menezes, A.; Oorschot, P.; Vanstone, S.: *Handbook of applied Cryptography*, CRC Press (1996)
- [14] PKCS #5 v2.0: *Password-Based Cryptography Standard*, RSA Laboratories (1999)