Technical recommendations on Cryptographic Mechanisms for IT and Security Personnel

EXECUTIVE SUMMARY

The comprehensive objective of SEISMED (a Secure Environment for Information Systems in MEDicine) was to elaborate a consistent, harmonized framework for medical data protection throughout Europe. The specific technical proposals of SEISMED are thus accompanied by a high level security policy which presents the underlying principles. This approach is consistent with the forthcoming European ITSEC activity.

SEISMED proposes a suite of cryptographic mechanisms in order to provide sufficient flexibility to meet the characteristic challenges of health care data processing:

- A long tradition of decentralized processing of health care data with multilateral and legitimate interests.
- Ultra high sensitivity of personal medical data whose disclosure might not be repairable by, e.g. smart-money.
- Long periods of time (up to 30 years) over which health care data must be archived in its original state.

A 20 man-month workpackage evaluated the pertinent cryptographic literature, other relevant EC-projects (RACE Integrity Primitives Evaluation project RIPE), and renowned conferences (IACR Crypto, IACR Eurocrypt, ACM Symposium on Theory of Computing, Symposium on the Foundations of Computer Science, IEEE Symposium on Research in Security and Privacy, etc.). The result is a cryptographic guideline which is presented by separate documents to three different target audiences.

Cryptographic guideline of SEISMED

Audience	Document Title	Content
Health Care Management	Guideline for Cryptographic Mechanisms —Health Care Management—	Odds and ends of cryptography: The use for health care IT-systems
IT-system end-users	Guideline for Cryptographic Mechanisms —IT-system end-users—	Odds and ends of cryptography: The benefits for IT-system end-users
IT and security personnel	Technical Recommendations for Cryptographic Mechanisms —IT and security personnel—	Suite of proposed cryptographic mechanisms

Health Care Management

This part addresses the management of a health care environment. While a rapidly evolving information technology often leaves the management confused, the management is responsible for keeping its health care environment working efficiently. To this end, *integrated IT systems* appear to be the ideal solution. This report identifies their specific risks which should be considered when deciding about new systems or upgrades of existing ones. This is even more important since indeed the management decides about IT-system installation, but in many cases the medical end-users like physicians are accountable for breaches of security of these systems. Hence, a management will only succeed in installing integrated IT-systems if it succeeds in inspiring the confidence of the end-users into these systems. Severe limitations of conventional security measures like passwords are identified and it is shown how these limitation can be overcome by applying cryptographic mechanisms. General aspects of integrating cryptography into existing applications are discussed.

IT-system end-users

This part addresses IT-system end-users, like physicians, medical staff, etc. who deal with sensitive medical data. Normally, these end-users are *personally* responsible for the medical data they input and process. Complementarily, they are also responsible if such data is modified or misused by other users. Hence, system end-users are particularly anxious about the risks which stem from the use of IT-systems. A conclusive introduction to the fundamental benefits of cryptography is provided which outlines how the identified risks can be reduced or eliminated.

IT and security personnel

This part addresses software and hardware designers and implementors who are responsible for the security of an IT-system. A suite of cryptographic mechanisms is proposed to be used by health care IT-systems.

First, the identified security requirements are mapped to cryptographic building blocks. Second, a few alternative cryptographic mechanisms are proposed to implement each building block. This two-step approach was found useful to make the document more readable and adaptable to future results in cryptologic research. In order to support the selection of mechanisms that comply with a given security policy, this report analyses explicitly how strong an adversary the proposed mechanisms resist. Two key applications in health care data processing are digital networks and databases. Specific proposals outline how these key applications can make use of the proposed cryptographic mechanisms.

Demonstrator

In order to demonstrate the functionality and efficiency of the recommended mechanisms, a software prototype (SECURE Talk) was built that protects data transferred through (linked) Apple Talk networks from unauthorised disclosure and undetectable modification. The demonstrator also provides an automatic key management for all cryptographic mechanisms. All results are documented in the document “Technical Recommendations for Cryptographic Mechanisms —IT and security personnel—”. SECURE Talk is available as a software application for Apple Macintosh from the author.

1 INTRODUCTION

Health care in Europe is characterized by a rapid development of information technology. More and more system components (hardware and software) for analysis, diagnoses, and treatment become increasingly efficient and integrated. The evolving integrated IT-systems are assumed to be widely distributed, potentially over many countries. Similarly, their control is assumed to be shared by many subjects like patients, physicians, medical staff, hospital administrations and managements, and national or even international institutions who want to communicate and cooperate. One of the characteristics of health care is that the different autonomous subjects who use the integrated IT-system have a common interest in such a system, but should and surely will stay autonomous. Every subject in health care has vital needs, and since hardly any subject will trust the entire IT-system, the system will only be credible and accepted if it is provably secure, i.e., if it provably respects the subjects' legitimate needs.

The notion of **security** can be formalized by specifying (i) a correct system behavior and (ii) an adversary model, both with respect to all subjects involved. Then a system is called secure if and only if it behaves correctly even if adversaries behave as maliciously as admitted by the adversary model.

The **correct system behavior**, (i), is often specified by some required services, which shall happen on demand, i.e., **availability**, and some forbidden services, which must not happen at all, i.e., **confinement**, (e.g., undetectable modification of data, repudiation of the origin of data, release of confidential data). Naturally, cryptographic means cannot enforce services, but they can prevent forbidden services or support to detect if they have happened. Hence, IT-systems are favorably equipped with cryptographic means since these means credibly support the subjects in keeping their individual interests of confinement.

The **adversary model** (ii) is usually specified by an **interaction model** and a **complexity model**. They describe the adversaries' capabilities with respect to interaction with the mechanism (e.g., which system components can be controlled in which way by adversaries?) and with respect to their computational complexity (e.g., which computational resources are available for adversaries?).

1.1 Which services of confinement are considered?

Four basic services of confinement are considered which can be achieved by means of cryptography:

(1) **confidentiality**,
(2) **data authentication** in the sense of **detectability of modification**,
(3) **entity and data authentication**, and
(4) **hashing**.

Hashing (4) is usually not counted among security services. But since it is sometimes useful in itself and is fundamental for the other three, it is included in the list. Obviously, there are other important aspects of security [ISO7498-2] like **access control**, and **integrity**. They comprise aspects of availability and of confinement. Access control means the possibility of access for authorized subjects as well as the restriction of access for unauthorized subjects. Integrity means that data is available and correct. The latter is covered by (2) and (3). A third

important security issue is **anonymity**. An individual might be anonymous during some action relative to all other parties performing the same action. To achieve such indistiguishable actions, synchronization protocols are needed. Since protocols are beyond this guideline, anonymity is not covered.

Each of the above services can be implemented by cryptographic mechanisms or a combination of several mechanisms. It is a characteristic trend in cryptology to provide relatively few elementary building blocks and to combine the more complex mechanisms from them. Advantages and dangers are obvious; cryptanalysis can focus upon the few building blocks, but combining two building blocks might insert new flaws.

All cryptographic mechanisms proposed are partitioned into four **cryptographic building blocks**: encipherment, message authentication codes, digital signatures, and hash mechanisms in order to provide the respective services (1), (2), (3), and (4). Fig. 1-1 gives an overview. All mechanisms of one cryptographic building block provide the same characteristic operations (e.g., encipher and decipher for encipherment). The operations achieve the confinement property of the respective service if they are used with suitable **cryptographic keys**. Consequently, individual interests of confinement can be achieved by using individual cryptographic keys.

Note: In principal, for different mechanisms different keys should be generated and used! Nevertheless, this guideline sometimes calls them by the same name. For all operations of Fig. 1-1 the key variables k, k' are used with the understanding that they are local parameters, NOT global constants!

services of confinement	cryptographic building block	**characteristic operations**	**examples of specific mechanisms**
confidentiality	encipherment	$c := encipher(k, m)$, $m := decipher(k', c)$	DES, RSA (with redundancy predicate)
detectability of modification	message authentication code (MAC)	$t := code(k, m)$, $check(k', m, t)$	Carter Wegmann Codes, DES-PCBC
non-repudiation	digital signature	$s := sign(k, m)$, $verify(k', m, s)$	GMR, RSA (with hash function)
collision-free hashing	hash mechanism	$hash(k, m)$	Iterated hash mechanisms

Fig. 1-1 Services and cryptographic building blocks

Also note that digital signatures and encipherment mechanisms are introduced as separate classes of mechanisms. There are, however, algorithms like RSA which are utilized to implement a mechanism of either class. This misled some authors into recommending that RSA encipherment could be used to achieve digital signatures in the following way:

$$s = sign(k', m) := decipher(k', m)$$
$$verify(k, m, s) \equiv (m \stackrel{?}{=} encipher(k, s))$$

Although this recommendation is algebraically correct, a naive implementation might lead to an insecure digital signature mechanism. Hence, the above equalities should not be employed for any secure mechanism design.

The mechanisms of one cryptographic building block may employ a mechanism of another building block. Some blocks systematically use other blocks, i.e., all of their mechanisms are implemented by the help of some mechanism(s) of the other building block. Some blocks, use other blocks only partially, i.e., only certain mechanisms are implemented by the help of some mechanism(s) of the other building block. Fig. 1-2 presents how cryptographic building blocks make use of one another.

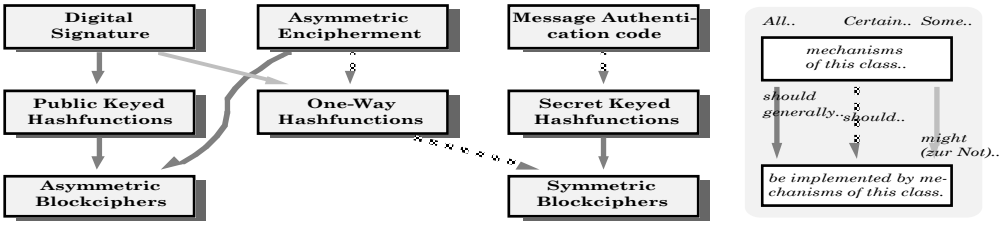

Fig. 1-2 How cryptographic building blocks make use of one another

1.2 What kind of adversaries are considered?

A measure for the security of a specific cryptographic mechanism is how strong an adversary it (provably or experimentally) resists. For each cryptographic building block the characteristic interaction models of adversaries are discussed. Basically, one distinguishes **passive** adversaries who can only receive certain output from the mechanism under attack from **active** adversaries who can also provide input to the mechanism. (E.g., ask questions.)

The three complexity models considered for adversaries are: unconditional, computational, and ad-hoc. They describe how complex a problem a respective adversary can **solve**, i.e. to compute a solution in time polynomial in the size of the problem instance [GaJo_79]. An unconditional adversary is able to solve all problems in **NP**. A computational adversary is only able to solve problems in **P**. An ad-hoc-adversary is unable to solve a particular problem (specified by the cryptographic mechanism under consideration). The only formal proofs of security without unproved assumptions are known for unconditional adversaries. Since $\mathbf{P} \neq \mathbf{NP}$ has not been proved so far, all formal proofs of security against computational adversaries rely on some complexity theoretic assumption. The better studied an assumption is, the more credible the proof appears. For example, the factorization assumption is well studied. It states that factoring certain integers is not in **P**. Against ad-hoc adversaries, security is not proved, but it is assumed entirely. These ad-hoc-assumptions are young (< 20 years) compared to the classical assumptions of complexity theory.

The security provided by the key of a cryptographic mechanism partially depends upon the choice of its **security parameter**(s). For example, a passive attack against the cryptographic key is an exhaustive search through the space of all possible keys. Hence, one of the most important security parameters is the cardinality of the space of possible

keys, which mainly determines the size of each key. Under further assumptions, the amount of time and money needed for a successful attack can be estimated from this security parameter.

1.3 How to read this guideline

After the relevant notation is introduced in chapter 2, chapters 3, 4, 5, and 6 propose specific cryptographic mechanisms for each of the four cryptographic building blocks. They are clearly arranged by a) proposing specific mechanisms[1] and advising the choice of their parameters, b) providing the necessary background information on cryptanalysis, c) showing the efficiency of the proposed mechanisms with respect to performance and memory requirements, and d) considering some aspects of standardization.

Field of interest	Section within chapters 3, 4, 5, 6
a) Proposals	1 Proposed mechanisms
	2 Choice of parameters
b) Background Information on cryptanalysis	3 Adversary models
	4 Cryptanalysis
c) Efficiency	5 Time and memory requirements
d) Standardization	6 Aspects of standardization

Section 1 presents the algorithms in a modular way beginning at the "top" and going down to the "bottom". Hence, the overall structure of the algorithms is presented first followed by more and more detail. The advantage of getting an overview before being flooded with mathematical detail hopefully outweighs the disadvantage of forward references. Moreover, forward references never point ahead more than a few sections.

In practice, one is looking for an "acceptable" trade off between security and efficiency (in terms of performance and budget). What "acceptable" means depends on, e.g., the results of risk analysis. To provide sufficient flexibility, for most mechanism classes *more* than one specific mechanism is provided. The guiding principles for the proposed suite of cryptographic mechanisms are:

- Mechanisms and their respective algorithms should have been published in the open literature and should have been proven to be secure. If no proof exists, the algorithms should have resisted several years of cryptanalysis by independent research.
- The recommendations should reflect the latest results of cryptologic research.
- One of the most secure, but still practical mechanism and
- one of the most efficient, but still secure mechanism should be recommended.

Chapter 7 deals with valuable combinations of some of the mechanisms introduced. There are two basic fields of application of cryptographic mechanisms, computer networks and databases. In computer networks usually numerous subjects like institutions and individuals have their own legitimate security interests and, thus, the interaction model of possible

---

[1] Throughout this report, the algorithmic detail of the recommended cryptographic mechanisms is completely referenced. For many of the recommended algorithms, BRUCE SCHNEIER provides descriptions and C-source code in [Schn6_93].

adversaries is highly decentralized. E.g., individuals might regard institutions or maintenance companies as adversaries. Thus, all the subjects have a need to exchange cryptographic keys. Chapter 8 proposes a key management for medical computer networks. In computer databases, usually the interaction model is more centralized. E.g., the operating system or the data base management system are regarded as adversaries with respect to confidentiality. Hence, sensitive data might be enciphered. Chapter 9 summarizes the theoretical limits of this approach and proposes some feasible solutions.

2 NOTATIONS

The following notations are used throughout this report.

iff …if and only if…

$\mathbb{N}$ denotes the set of positive integers {1, 2, …}

$\mathbb{N}_0$ denotes the set of non-negative integers {0, 1, 2, …}

$\mathbb{Z}$ denotes the set of integers {0, ±1, ±2, …}

$\mathbb{Z}_m$ $= \mathbb{Z}/m\mathbb{Z}$ denotes the set of residues modulo m

$\mathbb{Z}_m^*$ denotes the multiplicative group of $\mathbb{Z}_m$.

QR_m $= \{r^2 \mid r \in \mathbb{Z}_m^*\}$ the set of quadratic residues modulo m.

$\mathbb{P}$ $= \{2, 3, 5, 7, \ldots\}$ denotes the set of all primes.

$\mathbb{P}_{hard}$ denotes the set of primes p for which $p-1$ contains at least one big factor (i.e., $\geq p^{2/3}$) [Gord_85, BrDL_93].

$T_{l,\delta,\Delta,P,Q} = \{(p,q,m) \in \mathbb{N}^3 \mid \; p \in P \cap \mathbb{P}_{hard}, q \in Q \cap \mathbb{P}_{hard}, m = pq, len(m) = l,$
$\delta \leq |\text{len}(p) - len(q)| \leq \Delta \}$

denotes the set of integer triples such that the third component, the product of the first and second component, is l bit long and infeasible to factor if restricted to time polynomial in l.

$x :\in_R A$ denotes the random choice of an element x from the set A where any possible element is chosen with equal probability.

$x \leftarrow a()$ denotes the random choice of an element x by some indeterministic algorithm a.

$a \bmod n$ denotes the least non-negative rest of a division a by n, where $a \in \mathbb{Z}$, $n \in \mathbb{Z} \setminus \{0\}$.

$a \operatorname{div} n$ denotes integer division: $a \operatorname{div} n := \frac{a - a \bmod n}{n}$

$gcd(a, b)$ denotes the greatest common divisor of two integers a and b.

$cra(a \pmod n, b \pmod m)$ denotes the chinese remainder algorithm, i.e., the unique integer $c \in \{0, .., n \cdot m - 1\}$ such that $c = a \pmod n$ and $c = b \pmod m$.

$\oplus$ denotes the bitwise XOR operator. $\oplus: \{0,1\}^i \times \{0,1\}^i \rightarrow \{0,1\}^i$, for every $i \in \mathbb{N}$.

ε denotes the empty binary sequence.

$|$ denotes the operator for concatenation of binary sequences.

$len(x)$ denotes the function which returns the length of the binary sequence $x \in \{0,1\}^*$ in bit.

$substr(s, i, j)$ returns the j bit substring of the binary sequence s starting at the i. bit.

0xA1F denotes a constant in hexadecimal form. I.e., the number ((10•16)+1)•16)+15

The technical recommendations throughout this guideline comprise some **algorithms**. The algorithms are notated mathematically rather than in a certain programming language. Where appropriate a PASCAL-like syntax is used. Algorithms can be roughly specified by a mapping which presents their input and output domains. E.g.,

$a: A \rightarrow B$ denotes an algorithm a that takes one input from the set A and indeterministically outputs an element from domain B. Sometimes, not only one algorithm a, but a whole family a_F, $F = \{f, g\}$ of algorithms with equal domains is considered. If the domains do not coincide completely, but certain algorithms require inputs that others do not, those inputs are called optional and are enclosed in curly brackets.

$a_F: \{A\} \rightarrow B$ denotes a family of algorithms with optional input from A, i.e., not all of them take an input from A.

E denotes a block encipherment mechanism (chapter 3). It consists of two functional algorithms for encipherment and decipherment:

$$enc_E: EKey_E \times PB_E \rightarrow CB_E,\ dec_E: DKey_E \times CB_E \rightarrow PB_E$$

$EKey_E$, $DKey_E$ denote the domain of possible encipherment keys and decipherment keys, $PB_E = \{0,1\}^\beta$, $CB_E = \{0,1\}^\chi$ denote the domain of possible blocks of plaintext and ciphertext, respectively. The block lengths β and χ need not be the same. The third algorithm indeterministically generates pairs of matching encipherment and decipherment keys for E:

$$(ek, dk) \leftarrow keygen_E(\kappa), \text{ where } ek \in EKey_E,\ dk \in DKey_E$$

Generally, κ determines the cardinality of the domain $DKey_E$ of decipherment keys of E. ($|DKey_E| = 2^\kappa$)

D denotes a core authentication mechanism. It consists of two functional algorithms for producing and checking authentication tags:

$$ecd_D: EKey_D \times MB_D \rightarrow T_D,\ chk_D: CKey_D \times MB_D \times T_D \rightarrow \{\text{TRUE, FALSE}\}$$

$EKey_D$, $CKey_D$ denote the domain of possible encoding keys and checking keys, MB_D, T_D denote the domain of possible messages and tags, respectively. The third algorithm indeterministically generates pairs of matching encoding and checking keys for D:

$$(ek, ck) \leftarrow keygen_D(\kappa), \text{ where } ek \in EKey_D,\ ck \in CKey_D$$

Generally, κ determines the cardinality of the domain $EKey_D$ of encoding keys of D. ($|EKey_D| = 2^\kappa$)

G denotes a core signature mechanism. It consists of two algorithms for producing and verifying signatures:

sig_G: $SKey_G \times MB_G \rightarrow S_G$, ver_G: $VKey_G \times MB_G \times S_G \rightarrow$ {TRUE, FALSE}

$SKey_G$, $VKey_G$ denote the domain of possible signature keys and verification keys, MB_G, S_G denote the domain of possible messages and signatures, respectively. The third algorithm indeterministically generates pairs of matching signature and verification keys for *G*:

$$(sk, vk) \leftarrow keygen_G(\kappa), \text{ where } sk \in SKey_G, vk \in VKey_G$$

Generally, κ determines the cardinality of the domain $SKey_G$ of signature keys of *G*. ($|SKey_G| = 2^{\kappa}$)

In order to support the intuition, some algorithms will be presented graphically as well. This is particularly useful to clarify the flow of data between algorithms hosted at separate entities or the modular structure of an algorithm which itself uses other algorithms. Square and rounded boxes indicate deterministic and indeterministic behaviour, respectively.

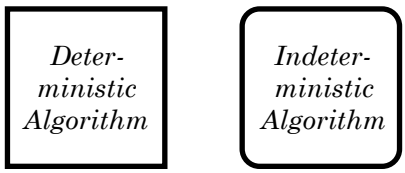

Fig. 2-1 Graphical representation of deterministic and indeterministic algorithms

3 ENCIPHERMENT MECHANISMS

Informally, the purpose of an **encipherment mechanism** (also called cryptosystems or simply ciphers) is to map binary sequences of "plaintext" to corresponding binary sequences of "ciphertext" in such a way that only someone who knows the decipherment key can efficiently recover the input sequences (plaintexts) from the output sequences (ciphertexts). In order to achieve this property, the encipherment operation is parametrized by an encipherment key whereas the decipherment operation is parametrized by a decipherment key. Let $EKey_E$, $DKey_E$ be respective domains of encipherment keys and decipherment keys, and let *P* and *C* be respective domains of possible plaintexts and ciphertexts. Then, an encipherment mechanism (sometimes also called **stream encipherment mechanism**) provides two characteristic operations:

$$encipher_E: EKey_E \times P \rightarrow C, decipher_E: DKey_E \times C \rightarrow P$$

and an indeterministic operation

$$(ek, dk) \leftarrow keygenerate_E(\kappa), \text{ where } ek \in EKey_E, dk \in DKey_E$$

which on input a security parameter κ, outputs a pair (ek, dk) of matching encipherment and decipherment keys. An encipherment mechanism yields for each such matching pair (ek, dk) and every input $p \in P$

$$decipher_E(dk, encipher_E(ek, p)) = p$$

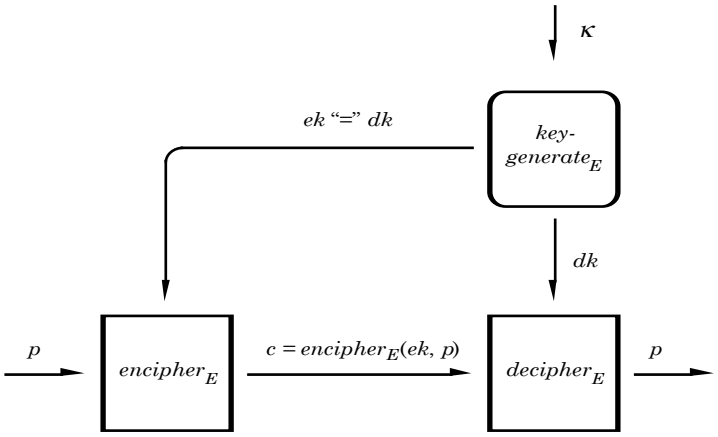

Fig. 3-1 Flow diagram for encipherment mechanisms

Fig. 3-1 gives the flow diagram for encipherment mechanisms. Deterministic algorithms are indicated by square boxes whereas indeterministic algorithms are given by capped boxes. In the sequel, these boxes will be refined to a certain extent. According to Fig. 3-1 refinements of encipherment and decipherment are shown on the left hand side and right hand side, respectively.

An encipherment mechanism is called secure if it resists at least a **total break**, i.e., if it is infeasible for an adversary to figure out the decipherment key. Sometimes, however, one will not even tolerate weaker kinds of breaks like the following in order of decreasing severity:

universal break: figure out an efficient decipherment algorithm which is functionally equivalent to $decipher_E$,

selective break: figure out the corresponding plaintexts to some particular ciphertexts chosen by the adversary,

existential break: figure out at least one pair of corresponding plaintext and ciphertext, no matter what the ciphertext is. This goal is trivial for asymmetric encipherment systems.

Note: The security of a well-designed encipherment mechanism does NOT rely upon the secrecy of its algorithms *encipher* and *decipher*, but solely upon the secrecy of its decipherment keys.

Encipherment mechanisms are partitioned into **symmetric encipherment mechanisms** (also called conventional encipherment mechanisms), **asymmetric encipherment mechanisms** (also called public key mechanisms) and **hybrid encipherment mechanisms**. For symmetric mechanisms it is feasible to compute the decipherment key from a given encipherment key, whereas for asymmetric mechanisms it is not. Hence, the encipherment key and the decipherment key of a symmetric mechanism are understood to be basically the same; they are both called a **secret key**. Those of an asymmetric mechanism

are understood to be significantly different; the former is called **public key** whereas the latter is called **private key**. Hybrid mechanisms provide the advantages of asymmetric encipherment (lean key management) and those of symmetric encipherment (efficiency).

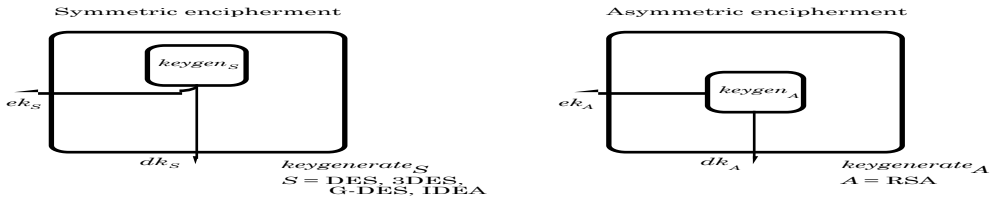

Fig. 3-2 Flow diagram for key generation for symmetric and asymmetric encipherment

It is proposed to employ **iterated encipherment mechanisms**. They are constructed by a generic iteration of a **block encipherment mechanism**. Symmetric and asymmetric iterated encipherment is reasonably achieved by a suitable iteration of a symmetric or asymmetric block encipherment, respectively. Iterated encipherment mechanisms only take input sequences whose length is a multiple of the block length of their respective block encipherment mechanism. Hence, input sequences of arbitrary length first have to be padded before they can be fed to the proposed iterated encipherment mechanisms.

3.1 Proposed mechanisms

The following proposals are useful compromises; they are not the desired optimum. This is because no encipherment mechanism is known so far which is provably secure and highly efficient. Hence, the best advice is to choose a compromise which meets one's requirements as close as possible. This is the reason why several mechanisms are proposed.

Chapter 3.1.1 presents hybrid encipherment which in its turn requires both asymmetric and symmetric encipherment. Chapter 3.1.2 describes the generic iteration for asymmetric and symmetric encipherment and the **input padding**. Chapter 3.1.3 proposes a classical **mode of operation** by which the partial results of the iteration are achieved. Finally, chapter 3.1.4 proposes five symmetric block encipherment mechanisms, and chapter 3.1.5 proposes an asymmetric block encipherment mechanism. The security of the former five bases on cryptographic intuition and experience. The security of the latter rests on the RSA-assumption which is not equally well investigated as the factorization assumption.

Fig. 3-3 proposes which block encipherment mechanisms (first column) should be combined with which auxiliary mechanisms (right columns).

block encipherment mechanism *E*	**padding**	**redundancy predicate**	**mode of operation**
DES, 3DES, G-DES IDEA	*enpad* *depad*	– (identity)	*enCBC, enPCBC* *deCBC, dePCBC*
RSA	*enpad* *depad*	$redins_{DES}$ $redchk_{DES}$	*enCBC, enPCBC* *deCBC, dePCBC*

Fig. 3-3 Proposal how to combine block encipherment with auxiliary mechanisms

3.1.1 Hybrid encipherment mechanisms

Definition: Let A, S be a pair of an asymmetric and a symmetric encipherment mechanism. Then, a **hybrid encipherment mechanism** is defined by the three operations:

$$encipher_{A,S}: EKey_A \times \{0,1\}^+ \rightarrow CB_A \times \{0,1\}^+$$

$$encipher_{A,S}(ek_A, p) := (encipher_A(ek_A, dk_S), encipher_S(ek_S, p))$$

where $(ek_S, dk_S) \leftarrow keygen_S$ is some randomly generated pair of keys.

$$decipher_{A,S}: DKey_A \times CB_A \times \{0,1\}^+ \rightarrow \{0,1\}^+$$

$$decipher_{A,S}(dk_A, c_1, c_2) := decipher_S(dk_S, c_2)$$

where $dk_S := decipher_A(dk_A, c_1)$.

$$keygenerate_{A,S} = keygenerate_A$$

Fig. 3-4 gives the flow diagram for hybrid encipherment and decipherment

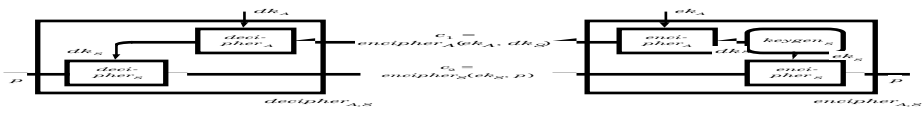

Fig. 3-4 Flow diagram for hybrid encipherment and decipherment

The two components c_1, c_2 of the ciphertext could, for example, simply be concatenated since the first component is of fixed length and is, thus, easily identified by the receiver of the ciphertext.

3.1.2 Iterated encipherment mechanisms

Definition: Let E be some block encipherment mechanism with PB_E, CB_E, $EKey_E$, $DKey_E$ its domains of plaintexts, ciphertexts, encipherment keys, and decipherment keys. Then, an **iterated encipherment mechanism** is defined by two functional operations

$$encipher_E: EKey_E \times P \rightarrow C$$

$$encipher_E(ek, p) := cb_1|cb_2|\ldots|cb_n,$$

$$\text{for all } i \in \{1, .., n\} \text{ let } cb_i := enmode(ek, pb_i, pb_{i-1}, cb_{i-1}), \text{ and } cb_0 := 0$$

where $pb_1|pb_2|\ldots|pb_n := enpad(\beta, redins_E(p))$, $pb_i \in PB_E$, and

$$decipher_E: DKey_E \times C \rightarrow P$$

$$decipher_E(dk, c) := redchk_E(depad(\beta, pb_1|pb_2|\ldots|pb_n)),$$

$$\text{for all } i \in \{1, .., n\} \text{ let } pb_i := demode(dk, cb_i, cb_{i-1}, pb_{i-1}), \text{ and } pb_0 := 0$$

where $cb_1|cb_2|\ldots|cb_n := c$, $cb_i \in CB_E$, dk some decipherment key for E, and β the length of plaintext blocks of E according to dk. The padding and depadding function (*enpad*, *depad*) are defined below. Some mechanisms require to distinguish valid from invalid plaintexts. The rationale behind it is to prevent an adversary from learning the plaintexts of arbitrary

ciphertexts by the help of a victim. It is proposed to apply the redundancy predicate defined in chapter 3.1.3. The mode of operation is proposed to be cipher block chaining defined in chapter 3.1.3 or plain cipher block chaining defined in chapter 7. Fig. 3-5 gives the flow diagram for a mode of operation during encipherment and decipherment.

Fig. 3-5 Flow diagram for splitting a stream of data into blocks and processing the blocks by a mode of operation

Third, the key generation is done by the corresponding indeterministic operation of E:

$$keygenerate_E = keygen_E$$

where $(ek, dk) \leftarrow keygen_E(\kappa)$, outputs a pair (ek, dk) of matching encipherment and decipherment keys $ek \in EKey_E$, $dk \in DKey_E$ on input a security parameter κ.

Definition: Padding and **depadding** are defined as follows:

$$enpad: \{1, 2, .., 15+16^4\} \times \{0,1\}^+ \rightarrow \{0,1\}^+$$

$$depad(\beta, s) := s|\underbrace{00\ldots0}_{v}|d_3|d_2|d_1|d_0, \text{ where}$$

$v = (b - 16 - len(s)) \bmod \beta$
$(d_3d_2d_1d_0)_{16}$ be the hexadecimal representation of v, i.e. $v = \sum_{i=0}^{3} d_i \times 16^i$.

Padding expands a sequence such that the length of the resulting sequence is a multiple of β, and such that the original sequence can easily be reconstructed. The sequence expansion is at most β+15 bit which poses the restriction $\beta \leq 15+16^4$ bit. If applied to block encipherment mechanisms this allows for block lengths up to 8193 byte which is far more than ever seems to be needed.

3.1.3 Redundancy predicate, mode of operation

A redundancy predicate works by inserting some redundant information into the plaintext before enciphering it (*redins*). After deciphering, the resulting plaintext is accepted as valid iff the redundancy check (*redchk*) succeeds. If *redchk* fails it returns an empty sequence instead of an invalid plaintext.

Definition: Let E be some block encipherment mechanism, χ be the length of blocks of ciphertext of E. H_E be a one-way hash mechanism (chapter 6.1.5) based on E. Then a stream oriented **redundancy predicate** is defined by two deterministic algorithms:

$$redins_E: \{0,1\}^+ \rightarrow \{0,1\}^+, redchk_E: \{0,1\}^+ \rightarrow \{0,1\}^*$$

$$redins_E(s) := s|hash_E(s)$$

$$redchk_E(t) := \begin{cases} s \\ \text{if } s' = hash_E(s); \varepsilon \\ \text{else} \end{cases}, \quad \text{where}$$

$s = substr(t, 1, len(t) - \chi); s' = substr(t, len(t) - \chi, \chi)$

The stream oriented insertion of redundancy expands the input sequence by an additional constant of χ bit.

Alternatively, the redundancy predicate of [ISO9796] can be used.

Definition: Let E be some block encipherment mechanism. Cipher Block Chaining (CBC) is defined as follows.

$enCBC: EKey_E \times PB_E \times PB_E \times CB_E \rightarrow CB_E$, $deCBC: DKey_E \times CB_E \times CB_E \times PB_E \rightarrow PB_E$

$enCBC(ek, pb, pb', cb) := enc_E(ek, pb \oplus cb)$, $deCBC(dk, cb, cb', pb) := pb \oplus dec_E(dk, cb)$

The flow diagram for the mode of operation is given by Fig. 3-6.

Fig. 3-6 Flow diagram for the mode of operation

Observe, that for CBC the block pb_{i-1} is not used at all and, hence, only the block cb_{i-1} has to be stored in every round. In other words, the blocks pb_{i-1} can be considered to be all zero which means that the addition mod 2^β can simply be omitted and the block cb_{i-1} is directly fed into the bitwise XOR.

3.1.4 DES, 3DES, G-DES, IDEA

In the sequel, the encipherment and decipherment, operations of one block of plaintext and the operation to generate a pair of matching keys will be denoted by

$$enc_E, dec_E, keygen_E \text{ for } E \in \{\text{DES, 3DES, G-DES, IDEA}\},$$

respectively. For the definition and implementation of the former two consult [Schn_93, pp. 219-243], for G-DES see [PfAß_90, PfAß_91], and for IDEA see [Schn_93, pp. 260-266]. Generation of keys is simply as follows: The domains of encipherment and decipherment keys are $EKey_E = DKey_E = \{0,1\}^\kappa \setminus \boldsymbol{WeakKeys}_E$, where the security parameter κ, and the set $WeakKeys_E$ are specific for each mechanism (chapter 3.2). The **key generation** ($keygen_E$) is by a uniformly random choice of an element k from the respective domain and by finally returning $(ek_S, dk_S) = (k, k)$ as a matching pair of keys.

DES [DES_77], 3DES [MeHe_81], G-DES [BiSh3_91], and IDEA [LaMa_91] are proposed as symmetric block encipherment mechanisms. The former three are published since 1977, the latter since 1990. All of them had already been thoroughly analyzed before their publication.

Even before it was standardized, the key size of DES (56 bit) was criticized as being too short. In order to achieve an increased effective key size **Triple-DES** (3DES) or **multiple encipherment** has been established in the literature [MeHe_81, KaRS_85, KaRS_88]. Although it is unlikely that the result of successive encipherment, each with a different key, can be achieved by only one encipherment with an appropriate key, this has never been proved to be impossible. 3DES encipherment and decipherment is defined as follows, with 3DES keys being twice the size of DES keys.

$$enc_{3DES}(ek_1|ek_2, s) := enc_{DES}(ek_1, dec_{DES}(ek_2, enc_{DES}(ek_1, s)$$

$$dec_{3DES}(ek_1|ek_2, s) := dec_{DES}(ek_1, enc_{DES}(ek_2, dec_{DES}(ek_1, s)$$

G-DES in the following means what [BiSh3_91] calls G-DES with the number q of parts per block restricted to $q = 2$ and the number n of rounds restricted to $n = 16$ [2]. Thus it is a generalization of DES in two respects.

Subkey independence: All 16 subkeys can be chosen independently from each other. This results in an effective key size of $16 \times 48 = 768$ bit. This generalization implies no extra run time cost of hard- or software enciphering and deciphering [BeFG_89, KFBG_90]. Hence, the generalization sticks to the design rule not to restrict the functionality of an implementation until this slows down its performance.

Design of S-boxes and permutations: The flow of data is not changed compared to DES, but all S-boxes and permutations can be chosen arbitrarily. (Of course S-boxes have to be designed carefully [AdTa1_90, Forr2_90, WeTa_86]!) This flexibility allows to reflect the latest results of cryptanalysis [DaTa_91]. Variable S-boxes do not affect the performance of software implementations and only slightly[3] reduce that of hardware implementations. Variable permutations also do not affect software implementations[4], but significantly reduce that of hardware implementations because much bigger matrices must be provided. Highly efficient software implementations can be obtained by the precomputation of working tables for S-boxes and permutations (chapters 3.5.1, 3.5.2). In order to allow for a uniform design of hard- and software implementations it is proposed to stick to the original permutations, but leave the S-boxes variable.

> **Note**: Variable S-boxes are only proposed as an option. The original S-boxes are proposed unless there are definite flaws detected.

IDEA is comparatively young in the field. Very poor progress in cryptanalysis of IDEA [Meie_94] encourages its use. Efficient software implementations of IDEA are available in the public domain.

3.1.5 RSA

In 1978, RONALD RIVEST, ADI SHAMIR, and LEONHARD ADELMAN published the first asymmetric block encipherment mechanism, but could not base its proof of security upon some thoroughly investigated complexity theoretic assumption (see chapter 3.3). After the

---

2 However, results of differential cryptanalysis (chapter 3.4) recommend to allow the number of rounds of G-DES to be $n \geq 16$.

3 This extra cost results from slower access to ROM than to RAM, because variable S-boxes have to be stored within RAM instead of ROM.

4 This is true as far as the structure of the “expanding permutation” E is restricted a bit [PfAß1_90].

names of its discoverers, the original mechanism has been referred to as **RSA**. The mechanism became quite popular and implementations of it are publicly available nowadays. For implementation consult [Schn_93, pp. 281-287]. The original mechanism has subsequently been improved by [QuCo_82, Denn_84, Damg_88].

Since pure RSA is a homomorphism it is proposed to enhance it by a redundancy predicate (Fig. 3-3).

Definition: **RSA block encipherment**

Let $\kappa \in \mathbb{N}$ be a security parameter, ω the computer word size in bit, $pb \in \{0,1\}^{\kappa-\omega}$ [5], $cb \in \{0,1\}^{\kappa}$, be a block of plaintext and a block of ciphertext, respectively, then encipherment and decipherment are defined as follows

$$enc_{\mathrm{RSA}}(ek, pb) := pb^{e} \pmod{n},\ dec_{\mathrm{RSA}}(dk, cb) := cb^{d} \pmod{n}$$

RSA keys are generated as follows:

$$(m, (p, q, d)) \leftarrow keygen_{\mathrm{RSA}}(\kappa)$$

where p, q are chosen randomly from the set *RSAPrimes*, such that

$m = p \times q \in T_{\kappa,10,20,RSAPrimes,RSAPrimes}$

$RSAPrimes = \{p \in \mathbb{P}_{hard} | p' \in \mathbb{P}_{hard}$ for at least one factor p' of $p-1$, $gcd(p-1, e) = 1\}$,

$d := e^{-1} \pmod{(p-1)(q-1)}$, $e = 2^{16} + 1$

Note: Implementations of RSA require a multiple precision integer arithmetic (MPIA) in order to process numbers of 100 up to 400 decimal digits depending on the security needed.

3.2 Choice of security parameter(s) and keys

There are at least three security parameters relevant for each block encipherment mechanism E proposed in chapter 3.1: the cardinality of the domain $EKey_E$ of encipherment keys, of blocks of plaintext PB_E, and of blocks of ciphertext CB_E. These cardinalities are determined by the parameters κ, β, χ. Furthermore, for some mechanisms optional parameters are proposed. Fig. 3-7 compares the proposed block encipherment mechanisms with respect to their security parameters. Constant parameters are just indicated, whereas for variable parameters a proposed range is given. Before picking some parameter from its proposed range chapter 3.4 should be consulted.

E	κ [bit]	$WeakKeys_E$	optional param.
DES	56	[DaPr_89, 65-66]	S-boxes
3DES	112	s.a.	S-boxes
G-DES	768	s.a.	S-boxes
IDEA	128	[DaGV1_94]	–

[5] The enciphering performance is the better the smaller ω is, but data is handled most easily if ω is the bit size of the smallest entity addressable by the host machine [Zim1_86]. Hence, ω should be the computer word size, usually 8, 16, or 32 bit.

RSA	{512, .., 1024}	–	–

Fig. 3-7 Security parameters of the proposed block encipherment mechanisms

3.3 Adversary models

Attacks are called **active** if the adversary is able to use the victim as an oracle by asking him questions and receiving his answers (typically without the victim being aware that he his under attack). Otherwise, they are called **passive**. Passive attacks against an encipherment mechanism can be classified into

ciphertext-only attacks: the adversary only gets to know some ciphertexts C_1, C_2… and is to infer the corresponding plaintexts.

known-plaintext attacks: the adversary additionally gets to know some pairs (p_1, c_1),(p_2, c_2),… of corresponding plain- and ciphertexts. The C_1, C_2… are pairwise different from the c_1, c_2…

Furthermore, active attacks are classified into:

chosen-plaintext attacks: the adversary performs a known-plaintext attack except that he himself chose the plaintexts p_1, p_2… before. This attack is trivial in the case of an asymmetric encipherment mechanism because its enciphering operation is publicly known.

(directed/adaptive) chosen-ciphertext attacks: the adversary performs a known-plaintext attack, but, before, he was able to choose the ciphertexts c_1, c_2,… If he had to choose all of them in advance the attack is called directed. If he could have chosen them one after the other depending on the plaintexts he obtained before the attack is called adaptive.

3.4 Cryptanalysis

The fundamental results are that symmetric encipherment mechanisms provably secure against adaptive chosen-ciphertext attacks require a key domain as large as the domain of plaintexts. (An example is the one-time pad.) Unfortunately, keys of that length are impractical for most applications. Asymmetric block encipherment mechanisms provably secure against chosen-ciphertext attacks can have keys of practical length, but so far only inefficient mechanisms are known. For example, [NaYu_90] requires non-interactive zero-knowledge proofs.[6]

Hence, for practical purposes, one can only apply symmetric block encipherment mechanisms based upon some highly chaotic function. Since their security rests upon some ad-hoc assumption their credibility relies on experience, i.e., years of unsuccessful cryptanalytic attempts. Similarly, one can only apply asymmetric block encipherment mechanisms based upon complexity theoretic assumptions less thoroughly investigated than the classical factorization assumption or the discrete logarithm assumption.

---

6 Note that [BlGo_85] is only secure against passive attacks. It is totally breakable by an adaptive chosen-ciphertext attack.

3.4.1 DES, 3DES, G-DES

DES, 3DES, and G-DES are chosen since they are well known, thoroughly cryptanalyzed, available, and permit relatively high enciphering rates implemented in both hard- or software.

No passive attack has come up that performs a total break of the full 16-rounds of DES, 3DES, or G-DES with significantly less effort than exhaustive search through the key space. [ChEv_86, BiSh3_91] attacked DES with less than 16 rounds, but were, hence, of theoretical interest only. **Differential cryptanalysis** [Bish_90, BiSh3_91] provides chosen-plaintext attacks that totally break DES up to 16 rounds. ELI BIHAM and ADI SHAMIR showed how this technique can be adapted to a lot of variants of DES. One specialized and improved version of this attack breaks the full 16-round DES [BiSh4_91] at the adversary's cost of about $2^{37.2}$ trial encipherments and a minimum amount of space. As input it requires the ciphertexts corresponding to about 2^{47} plaintexts chosen by the adversary in advance. FEAL and N-Hash are analyzed in [BiSh_91] and Snefru, Khafre, REDOC-II, LOKI and Lucifer are analyzed in [BiSh_92, BKPS_93]. Since these publications, a lot of proposals came up to improve the above mentioned variants of DES. Nevertheless, the original DES seems to be one of the hardest variants in the light of this latest cryptanalytic technique. In turn, multiple encipherment appears to be even less vulnerable by differential cryptanalysis than single encipherment DES.

Whether a chosen-plaintext attack, which requires at least 2^{47} trial plaintexts, is indeed a threat for a particular application, very much depends on other circumstances. In general, the more a cryptographic device enciphers using the same key without asking why or for whom, the more serious a chosen-plaintext attack against its key is.

So far, no attack has come up that aims at a universal break of DES, 3DES, or G-DES, but such an attack might be possible. The criteria used for S-boxes have not been published completely, neither have the complete criteria for their actual choice. Although [Copp_92] came up with the most important ones, the skeptics were never convinced that no trapdoor was built into the S-boxes of DES.

3.4.2 How powerful is differential cryptanalysis?

Let *te* denote the time of the DES encipherment of one plaintext block (64 bit). An active attack against full 16 round DES can be performed in $2^{37.2}te$ adversary time plus $2^{47}te$ victim time (for the encipherment of 2^{47} chosen plaintexts). This results in a ratio of

$$\frac{dc_{\mathrm{DES}}}{es_{\mathrm{DES}}} = 2^{47-56} = 2^{-3} \approx 1.95 \times 10^{-3} \tag{3.1}$$

compared to the complexity of the best known passive attack (exhaustive search through the key space).

An active attack against G-DES with the S-boxes DES and 16 independent subkeys (48 bit each) requires the ciphertexts of 2^{59} chosen plaintexts and $2^{61}te$ adversary time, which is about $2^{61-768} = 2^{-707}$ of the complexity of the best known passive attack (exhaustive search through the key space of size 2^{768}).

[GaOu_91] estimates the time and cost to break DES. It assumes that an exhaustive search (*es*-attack) of the key space costs $es_{\mathrm{DES}} = 2^{56}te$ time. Figures 3-8 to 3-11 compare the costs of an *es*-attack to an active attack by differential cryptanalysis (*dc*-attack). A dc-

attack costs $2^{37.2}te$ time for DES ($2^{61}te$ time for G-DES) and additionally the encipherment of about 2^{47} (2^{59}) plaintexts chosen by the adversary in advance. This results in an approximate overall cost of $dc_{DES} = 2^{47}te$ ($dc_{G\text{-}DES} = 2^{61}te$) time. Although the latter costs might be decreased in near future by applying the new ideas of [BiSh4_91] this report conservatively assumes $dc_{G\text{-}DES} = 2^{61}te$. Fig. 3-8 and Fig. 3-10 present the estimations of [GaOu_91] for DES. Fig. 3-9 and Fig. 3-11 extrapolate those estimations to a *dc*-attack by applying the ratio (3.1)

speed of processors	**# processors required to run an *es*-attack against DES within:**			
(Key tests / s)	**1 year**	**1 month**	**1 week**	**1 day**
1 million	2,300	27,400	119,200	834,000
2 million	1,150	13,700	59,600	417,000
4 million (1990)	600	6,850	29,800	208,500
32 million (1995)	75	850	3,700	26,100
256 million (2000)	9	107	500	3,300

Fig. 3-8 Number of processors required to run an *es*-attack against DES

Analogous results for G-DES are not known to be published. A conservative estimation should calculate

$$es_{G\text{-}DES} = 2^{768}te = 2^{768-56}es_{DES} = 2^{712}es_{DES}$$
$$dc_{G\text{-}DES} = 2^{61}te = 2^{61-56}es_{DES} = 2^{5}es_{DES}$$

Fig. 3-8 and Fig. 3-9 show the number of processors required for breaking DES by exhaustive search (ciphertext only attack) or by differential cryptanalysis (chosen-plaintext attack), respectively.

speed of processors	**# processors required to run a *dc*-attack against DES within:**			
(Key test equivalents / s)	**1 year**	**1 month**	**1 week**	**1 day**
1 million	5	54	233	1,629
2 million	3	27	117	815
4 million (1990)	2	14	59	408
32 million (1995)	1	2	8	51
256 million (2000)	1	1	1	7

Fig. 3-9 Number of processors required to run a *dc*-attack against DES

speed of processors	**investment [$1,000] required to run an *es*-attack on DES within:**			
(Key tests per second)	**1 year**	**1 month**	**1 week**	**1 day**
4 million (1990)	129.0	1,532	6,664	46,622

32 million (1995)	52.0	600	2,611	18,265
256 million (2000)	10.3	117	510	3,580

Fig. 3-10 Investment required to run an *es*-attack against DES

speed of processors	**investment [$1,000] required to run a *dc*-attack on DES within:**			
(Key test equivalents / s)	**1 year**	**1 month**	**1 week**	**1 day**
4 million (1990)	0.252	2.992	13.016	91.059
32 million (1995)	0.102	1.172	5.100	35.674
256 million (2000)	0.020	0.229	0.996	6.992

Fig. 3-11 Investment required to run an *es*-attack against DES

Fig. 3-10 and Fig. 3-11 show the estimated investment required to break DES by exhaustive search or by differential cryptanalysis, respectively. “A cost of $25.00 per processor was assumed for the cheapest available technology, rising by a factor of ten for each 'generation' of technology (roughly, five years newer and ten times faster), and decreasing by a factor of 10 (before deflation) every five years. The base figure of $25.00 includes the cost of designing boards, control software and manufacturing. […] The results represent a guess with an error of perhaps 50%.” (See [GaOu_91] for the exact basis of estimation.)

3.4.3 IDEA

Promising candidates of a block encipherment mechanisms resisting differential crypt-analysis are the **MARKOV ciphers** with more than 4 rounds. One was published as a “Proposed Encryption Standard” (PES) or “International Data Encryption Algorithm” (IDEA) [LaMa_91, LaMa2_91]. It provides a key size of 128 bit.

3.4.4 RSA

RSA is proposed since it is well known, thoroughly cryptanalyzed, and available. Active attacks that aim at a total break are not known. Passive attacks that aim at a total break are exhaustive search through the secret key space and factorization of the public modulus. The former costs exponential time (in the security parameter $\kappa = \ln(m)$) and is, thus, less economical than factorization, which can be done in subexponential time. One of the most efficient known algorithms for factorization is the multiple polynomial quadratic sieve (MPQS) which succeeds in

$$L(m) = \exp\left(\left(1 + o(1)\right)\sqrt{\ln(m)\ln\ln(m)}\right) \text{ steps}$$

The number field sieve (NFS) succeeds faster for integers of a special form [LeMa1_90, LeLe_90].

Other passive attacks have been considered by [SiNo_77, Herl_78, Berk_82], but were found to be impractical [Rive_78, Rive_79, WiSc_79, Berk_82] if the prime factors p, q and the public exponent e are chosen according to chapter 3.1.

The security of RSA rests never been shown to be equivalent to some classical complexity theoretic assumption, e.g., factorization of integers. Hence a universal break might be possible, but is not known.

Pure RSA is an isomorphism from the domain of blocks of plaintexts onto the domain of blocks of ciphertexts. Thus, a selective break can be accomplished by a chosen-ciphertext attack [Denn_82]. To overcome that weakness, a redundancy predicate is strongly recommended for RSA. Indeed, [ACGS_88] have shown that figuring out certain particular bits of plaintext is as hard as figuring out the whole plaintext out of a given ciphertext.

3.4.5 How powerful is factorization

Basically there are two passive attacks aiming at a total break of RSA; namely, exhaustive search through the key space and factorization of the public modulus.

For a fixed size k of an RSA modulus the performance of exhaustive search can be roughly estimated based on the DES results and a comparison of the current state of DES and RSA hardware performances. In 1990 DES hardware achieved deciphering rates of roughly 200 times as fast as RSA hardware did. Given a similar hardware development for RSA as for DES the results of 3-8 and 3-10 may be extrapolated respectively.

The performance of factoring integers depends on the factorization algorithm actually used and on its (hardware) implementation [see BrOd_92]. The most efficient factoring algorithms are actually the multiple polynomial quadratic sieve (MPQS) and the number field sieve (NFS) [LeMa1_90]. So far the NFS is best suited for integers of special form. The complexity of MPQS is subexponential and thus beats a brute force attack against the modulus by exhaustive search. [LeMa1_90] report one of the best factorization results so far: They cracked a general integer of approximate length 330 bit in about 26 days. They distributed the execution over a network of mainframes.

Given a security policy and thereby a lower bound for the period of integrity of an RSA instance, it is hard to calculate an appropriate key size κ for the required instances. Uncertainty arises, for example, from unpredictable technical and algorithmic progress.

[Rive4_91] looks at a period of 25 years (which might be far too short for some medical applications). Within this period he assumes that no significant progress in factoring is achieved and that the computational power available per dollar constantly increases by 40% per year[7]. He further approximates the cost per **MIPS-year**[8] ($\approx 31.54 \times 10^{12}$ operations) to about $4.00. The following table Fig. 3-12 gives the resulting investment needed to factor an integer of size κ in 1992/2017 respectively.

It must be stressed, however, that the integrity of data in real applications depends at least as much on the security of the key management as on that of the encipherment mechanism itself. This includes distribution, storage and erasure of keys (cf. chapter 8).

modulus size κ [bit]	**number of operations [MIPS-year]**	**investment in 1992 [$]**	**investment in 2017 [$]**
512	2.12×10^6	8.48×10^6	1,696
700	19.57×10^9	78.27×10^9	15.65×10^6

7 For further assumptions see [Rive4_91]

8 Number of operations performed by a 1 MIPS processor during 1 year.

800	1.61×10^{12}	6.45×10^{12}	1.29×10^{9}
1000	5.55×10^{15}	22.21×10^{15}	4.44×10^{12}
1100	248.04×10^{15}	992.15×10^{15}	198.43×10^{12}

Fig. 3-12 Investment to factor a single RSA modulus according to [Rive4_91]

3.4.6 Valuation of the proposals

If high speed is of prior importance, DES or IDEA are recommended. If IDEA is not available, but active attacks have to be countered, 3DES is a good choice with respect to current expertise. Besides, DES, 3DES, G-DES, IDEA, and RSA there are a lot of other block encipherment mechanisms proposed in the literature. Some of them are sorted out because they were at least partially broken (e.g. encipherment mechanisms based on knapsacks, and most mechanisms based on Linear Feedback Shift Registers LFSRs etc.), some have turned out at least not to be significantly stronger than DES or 3DES in the light of differential cryptanalysis (e.g., FEAL, N-Hash, Snefru, Khafre, REDOC-II, LOKI, Lucifer), and some younger ones remain to be investigated further (e.g. encipherment mechanisms based on elliptic curves). In contrast to the "classical" DES and RSA which have now been analyzed for about 16 years, these younger mechanisms have not been cryptanalyzed to a similar extent. Consequently, they are not recommended here. But of course, further progress in cryptologic research might show that other encipherment mechanisms have benefits in security and practicality that make them worth being considered in the medical field.

If asymmetric encryption is needed RSA is recommended. If both, asymmetric behavior and high speed are required, hybrid encipherment is proposed.

3.5 Implementations and their performance

3.5.1 DES hardware implementations

Six of the fastest DES chips are presented in the Fig. 3-13. Most of them offer several modes of operation. The performances do not differ significantly between different modes as far as 64 bit per block are processed.

	availability	**technology [µm]**	**chip size [mm²]**	**modes of operation**	**clock [MHz]**	**bitrate [Mbit/s]**
[HoGD_85]	1985	3	$4.0 \times 4.0 = 16.0$	? [9]	?	20
DEP T7000A [AT&T]	1987	?	?	ECB, CBC, CFB, OFB	?	15.056
G-DES, M-DES [KFBG_90]	1988	2	77.32 (40% free)	ECB, CBC, CFB, OFB	10	40 [10]

[9] Question marks indicate a lack of exact information.

[10] This result is achieved internally upon the DES kernel of the chip. Externally a performance of 10 Mbit/s is achieved because access to the DES kernel is byte serial and asynchronous.

DES (G-DES) [VHVD_88]	1988	3 (CMOS)	5.0 × 5.0 = 25.0	ECB, CBC, CFB, OFB	16.7	20 [11]
AMD[12] [Stel_86, AMD_85]	1989	?	?	ECB, CBC, CFB	?	14
uti-maco [SAFE_90]	1990	?	?	?	?	18

Fig. 3-13 Performance of DES-Hardware

The implementations [KFBG_90, VHVD_88] not only allow to run DES, but also G-DES. The performances show that this generalization does not reduce the performance of these chips.

3.5.2 DES, G-DES software implementations

Some of the fastest known software implementations are summarized in Fig. 3-14. Note that the 2nd and 3rd are G-DES implementations, the former being a fine-tuned version for the Apple Macintosh, the latter a comparable version for the IBM-PC (8086 Code).

	availability	**supported processor**	**precomp. [Kbyte]**	**modes of operation**	**reference machine**	**bitrate [Kbit/s]**
DES [PfAß_90]	1990	680x0 x ∈ {0…4}	106	all [13]	Apple Mac IIfx [14]	1109
G-DES [PfAß_90]	1990	680x0 x ∈ {0…4}	106	all	Apple Mac IIfx	789
G-DES [Mühl_89]	1990	8086, 80x86 x ∈ {1…4}	106	all	Toshiba T5100 [15]	148.9
Rescrypt [Resc_91]	1991	8086, 80x86 x ∈ {1…4}	?	ECB	386 Clone [16]	280

Fig. 3-14 Performance of DES-Software

3.5.3 3DES, IDEA

A software implementation of *n*-encipherment can, according to [PfAß1_90], yield a performance of about

$$p(n) = \frac{p(1)}{0.7n + 0.3},$$

---

11 Bitrates of 32 Mbit/s are planned by integrating a complete controller and microprocessor upon one Chip.

12 AMD: Am9518, Am9568, Am28068, AmZ8086

13 These are: ECB, CBC, CFB, OFB, PCBC

14 Apple Macintosh IIfx (MC68030, 40 MHz, 32 Kbyte cache board, 80 ns RAM)

15 Toshiba T5100 (Intel 80386, 16 MHz)

16 Processor: Intel 80386, 33 MHz

where $p(1)$ denotes the performance of single encipherment DES. This is due to the fact that the trailing permutation of a preceding encipherment is neutralized by the leading permutation of the successive encipherment. Hence, only the leading permutation of the first encipherment and the trailing permutation of the last encipherment have to be performed. Thus, the performance of 3DES is $p(3) = 0.41 \times p(1)$

Latest benchmarks of software implementations of IDEA suggest that it performs by a factor of 1.5 or 2.0 faster than single DES on the same host.

3.5.4 RSA hardware implementations

Five of the fastest RSA chips are shown in Fig. 3-15. The bitrate refers to a general, pure deciphering (i.e. large secret exponent, no redundancy predicate). Using the Chinese remainder algorithm speeds up deciphering by almost the factor 4, but it is not known if the chips use this shortcut[17]. The last entry gives an impression of what can be implemented upon a Smart Card. (The number of bits processed per chip indicates how many chips have to be cascaded in order to process blocks of larger size.)

	avail-ability	**technology [µm]**	**chip size [mm^2]**	**#bits/chip**	**clock [MHz]**	**Enciphering rate [Kbit/s] (length of *m*)**
Cryptech [Bric_90]	1988	GateArray	?	120	14	17 (512)
[Sedl_88]	1989	5	4.8 × 5.0	780 [18]	?	195 (780)
VICTOR [OrSA_91]	1990	2	10.0 × 10.0	512	20	97 (512)
[VVDJ_90]	1990	2 (CMOS)	9.3 × 8.7	1024	25	8 (1024)
CORSAIR [QuWB_91]	1991	1.2 (CMOS)	1.7 × 1.7	512	6	0.34 (512)
Philips DX-Card	1992	?	1.7 × 1.7	512	8	40.96 (512) ?

Fig. 3-15 Performance of RSA-Hardware

The 3rd and 5th entry are prototypes that are not commercially available. More descriptions for RSA hardware can be found in [ORSP_87, Barr_87]. The former is an announcement for a chip the latter makes use of Motorola's digital signal processor DSP56200.

3.5.5 RSA software implementations

The performances of Fig. 3-16 refer to pure RSA decipherment (i.e. no redundancy predicate is used) at a modulus size of 512 bit.

A lot of other implementations are documented in the literature [BoRu_89, Comb_90, Jung_87, LiPo_91]. The one by P. COMBA is comparable to the above mentioned work by D. FOX, the other three are a good deal less efficient.

---

17 Often exact information about RSA hardware is held secret by designers and manufacturers

18 prime factors of the modulus are restricted to sizes of 340 bit and 440 bit, respectively.

	availability	supported processor	precomputation [Kbyte]	modes of operation	reference machine	deciphering rate [ms/512bit]
RSA	1993	680x0 x ∈ {0…4}	0	ECB, CBC	Apple [19] Quadra 950	100.0 (= 5.0 Kbit/s)
RSA [Fox_91]	1990	8086, 80x86 x ∈ {1…4}	0	ECB, CBC	386 Clone [20]	490[21] (= 1.05 Kbit/s)
Rescrypt [Resc_91]	1991	8086, 80x86 x ∈ {1…4}	?	ECB	386 Clone [22]	330.3 (= 1.55 Kbit/s)

Fig. 3-16 Performance of RSA-Software

3.6 Standardization

DES was standardized within the U.S. [DES_77] to be used for unclassified data. G-DES was not standardized in the U.S. No international standard exists for DES and G-DES.

Since 1986 the international standards organization (ISO) has deliberately not standardized particular encipherment mechanisms any more. Alternatively a register of such mechanisms is run according to the rules proposed within [ISO9979]. One of these rules is, that the register neither has to tell anything about how a mechanism works nor what level of security it provides [Pric_88, Pric_90]. For example, the European project RIPE - the RACE integrity primitive evaluation project has concentrated on integrity and authentication only.

RSA is not standardized internationally either. However there are de-facto standards [RSAD_91]. Also see [Kali3_91] for a nice overview.

Modes of operations for block ciphers are standardized in [ISO8372] and more generally in [ISO10116].

4 MESSAGE AUTHENTICATION CODES

Informally, the purpose of a **message authentication code** (MAC) is to prove that the real originator of a message is indeed the claimed one. However, the proof by an authentication code will only convince someone who trusts in every other entity who also knows the checking key for messages of the real originator. This includes the real originator himself. The reason is that anyone able to check a message authentication code is at the same time able to produce it. See also [Simm1_92] for an overview.

Mathematically, a message authentication code maps binary sequences to corresponding “tags” in such a way that

- only someone who knows the encoding key or checking key can efficiently produce and check tags.

---

19 Apple Macintosh Quadra 950 (MC68040, 30 MHz, 80 ns RAM)
20 Processor: Intel 80386, 33 MHz, 64 Kbyte cache board
21 8086 implementation: no support of 32-bit multiplication instruction (80386, 80486).
22 Processor: Intel 80386, 33 MHz

In order to achieve this property, the encoding operation is parametrized by an encoding key whereas the checking operation is parametrized by a checking key. Let *EKey*, *CKey* suitable domains of encoding keys and checking keys, and let *M* and *T* be the respective domains of binary sequences (messages) and tags. Then, a message authentication code provides two characteristic functional operations:

$$encode: EKey \times M \rightarrow T,\ check: CKey \times M \times T \rightarrow \{\text{TRUE}, \text{FALSE}\},$$

and an indeterministic operation

$$(ek, ck) \leftarrow keygenerate(\kappa), \text{ where } ek \in EKey, ck \in CKey$$

which on input a security parameter κ, outputs a pair (*ek*, *ck*) of matching encoding and checking keys. A message authentication code yields for each such matching pair (*ek*, *ck*) and every input $m \in M$

$$check(ck, m, encode(ek, m)) = \text{TRUE}$$

Fig. 4-1 gives the flow diagram for message authentication codes. Deterministic algorithms are indicated by square boxes whereas indeterministic algorithms are given by capped boxes.

A message authentication code is called secure if it resists at least a **total break**, i.e., if it is infeasible for an adversary to figure out the encoding key. Sometimes, however, one will not even tolerate weaker kinds of breaks like the following in order of decreasing severity [GoMR_88]:

universal break: figure out an efficient encoding algorithm which is functionally equivalent to $encode_D$,

selective forgery: forge a tag for some message chosen by the adversary,

existential forgery: forge a tag for any one message, even a random or non-sensical one.

> **Note:** The security of a well-designed message authentication code does NOT rely upon the secrecy of its algorithms *encode* and *check*, but solely upon the secrecy of its encoding and checking keys.

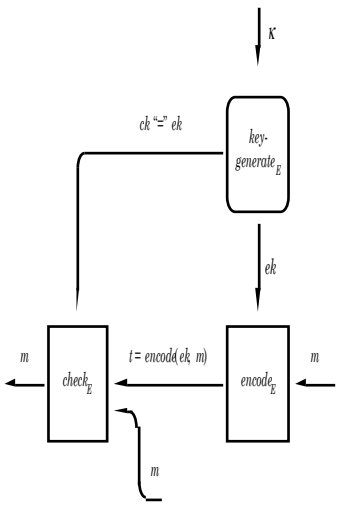

Fig. 4-1 Message Authentication Code

Message authentication codes are symmetric cryptographic mechanisms, i.e., it is easy to compute the matching encoding key from a given checking key. Hence, encoding keys and checking keys are secret keys.

Analogous to digital signature mechanisms (chapter 5), message authentication codes can be described as compositions of a **hash mechanism** (chapter 6) and a **core authentication mechanism**. This approach is adopted because it profits by independent cryptologic research on hash mechanisms. Furthermore, it allows to design message authentication codes more appropriately to meet specific needs.

4.1 Proposed mechanisms

The following proposals provide the desired optimum with respect to security: a provably secure message authentication code. Furthermore, an alternative mechanism is provided whose security rests on an ad-hoc assumption. Chapter 4.1.1 precisely introduces how message authentication codes are constructed from core authentication mechanisms (chapter 4.1.2) and hash mechanisms (chapter 6.1.2 and 6.1.3). Fig. 4-2 suggests combinations of the core authentication mechanism and hash mechanisms.

core authentication mechanism *D*	**hash mechanisms**
CW	sDES, sIDEA
CW	sCW1

Fig. 4-2 Proposal how to combine core authentication mechanisms with hash mechanisms

On the one hand, the core authentication mechanism CW provides unconditional security only if it is combined with a 2-universal hash mechanism like sCW1. On the other hand, it is highly efficient and may thus be combined with faster hash mechanisms like sDES or sIDEA.

4.1.1 Composition of message authentication codes

Definition: Let H, D be a hash mechanism and a core authentication mechanism. Then, a **message authentication code** is defined by the three operations:

$$encode_{H,D}: EKey_D \times HKey_H \times \{0,1\}^+ \rightarrow T_D$$
$$encode_{H,D}(ek, hk, m) := ecd_D(ek, hash_H(hk, m))$$

$$check_{H,D}: CKey_D \times HKey_H \times \{0,1\}^+ \times T_D \rightarrow \{\text{TRUE, FALSE}\}$$
$$check_{H,D}(ck, hk, m, t) := chk_D(ck, hash_H(hk, m), t)$$

where $hk \leftarrow keygen_H$ is some hashing key, and $(ek, ck) \leftarrow keygen_D$ is some pair of matching encoding, checking keys.

$$(hk, ek, ck) \leftarrow keygenerate_{H,D}(\kappa)$$

where the key generation works by choosing hk by applying $keygen_H$, and by *independently* choosing (ek, ck) by applying $keygen_D$.

4.1.2 CW

In 1981, LAWRENCE CARTER and MARC WEGMAN published a message authentication code provably secure against the strongest possible attack. After the names of its discoverers, the original mechanism is referred to as **CW**. See the original work [CaWe_79, WeCa_81].

Definition: CW core authentication

Let $\kappa \in \mathbb{N}$ be a security parameter, $mb \in \{0,1\}^\kappa$, $t \in \{0,1\}^\kappa$, be a message block and a tag, respectively, and B be an upper bound for the number of tags that can be produced by one encoding key. Encoding and checking signatures is defined as follows

$$ecd_{\mathrm{CW}}(ek, mb) := r_i \oplus mb,$$

$$chk_{\mathrm{CW}}(ck, mb, t) := r_i \oplus mb \stackrel{?}{=} t$$

where $ek = ck = (r_1, r_2, .., r_B)$. CW keys are generated as follows:

$$(r_1, r_2, .., r_B) \leftarrow keygen_{\mathrm{CW}}(\kappa, B)$$

where each r_i, $i \in \{1, 2, .., B\}$ is uniformly randomly chosen from $\{0,1\}^\kappa$.

4.2 Choice of parameters

There is one primary security parameter τ relevant for any core authentication mechanism D. τ determines the cardinality of the domain T_D of possible tags. Unconditionally secure MACs yield a forging probability of $2^{-\tau}$ and require a hashing key domain of cardinality at least $2^{-2\tau}$ [GiMS_74]. Some mechanisms provide additional or optional parameters. Fig. 4-3 provides a range of proposed values for each core signature mechanism. Before picking some parameter chapter 4.4 should be consulted.

D	τ [bit]	b [#tags]
CW	{50, .., 100}	> 0

Fig. 4-3 Security parameters proposed for the core signature mechanisms

4.3 Adversary models

Attacks are called **active** if the adversary is able to use the victim as an oracle by asking him questions and receiving his answers (typically without the victim noticing to be under attack). Otherwise, they are called **passive**. A passive attack against a message authentication code can be a

known-message attack: the adversary is aware of some pairs (m_1, t_1), (m_2, t_2),… of corresponding messages and corresponding tags.

Furthermore, active attacks are classified into:

directed-chosen-message attack: the adversary performs a known-message attack except that he himself chose the messages m_1, m_2… before.

adaptive-chosen-message attack: the adversary performs a directed-chosen-message attack, except that he did not have to choose all the messages m_1, m_2… in advance, but

rather could have chosen them one after the other depending on the tags he obtained before.

4.4 Cryptanalysis

Details about differential cryptanalysis of message authentication codes based on block ciphers like DES or IDEA can be found in [OhMa_93].

4.4.1 CW

The probability of a successful existential forgery of CW is $2^{-\tau}$. This holds even under the most serious attack (i.e. adaptive-chosen-message). The security is unconditional, i.e., it relies on no further assumptions.

4.5 Performance

For both message authentication codes the performances can be taken from the underlying hash mechanisms (chapter 6.5). This is because the MAC based on sDES or sIDEA does not need a core authentication mechanism at all, and the MAC based on sCW1 employs a highly efficient core authentication mechanism.

4.6 Status of standardization

DES based message authentication codes are standardized in [ISO9797, ISO8731].

5 DIGITAL SIGNATURE MECHANISMS

Informally, the purpose of a **digital signature mechanism** is to prove that the real originator of a message is indeed the claimed one. In contrast to authentication codes, the proof by a digital signature will convince everyone who only trusts in the real originator himself. In general, anyone able to verify a digital signature is NOT able to produce it.

Mathematically, a digital signature maps binary sequences to corresponding "signatures" in such a way that

- only someone who knows the signature key can efficiently compute signatures,
- anyone who knows the matching verification key can verify those signatures, but cannot produce a signature by himself.

Unlike the authentication codes (chapter 4), digital signatures can be employed for a non-repudiation service. To this end, independent "courts" must be provided which must have reliable access to the verification keys. This might be achieved by some registration process that every user participant has to pass.

In order to achieve the above properties, the signature operation is parametrized by a signature key whereas the verification operation is parametrized by a verification key. Let *SKey*, *VKey* be suitable domains of signature keys and verification keys, and let *M* and *S* be respective domains of binary sequences (messages) and signatures. Then, a digital signature mechanism provides two characteristic operations:

$$sign: SKey \times M \rightarrow S,$$

$$verify: VKey \times M \times S \rightarrow \{\text{TRUE}, \text{FALSE}\},$$

and an indeterministic operation

$$(sk, vk) \leftarrow keygenerate(\kappa), \text{ where } sk \in SKey, vk \in VKey$$

which on input a security parameter κ, outputs a pair (sk, vk) of matching signature and verification keys. A digital signature mechanism yields for each such matching pair (sk, vk) and every input $m \in M$

$$verify(vk, m, sign(sk, m)) = \text{TRUE}$$

Fig. 5-1 gives the flow diagram for digital signature mechanism. Deterministic algorithms are indicated by square boxes whereas indeterministic algorithms are given by capped boxes. In the sequel, these boxes will be refined to a certain extent. According to Fig. 5-1 refinements of verifying and signing are shown on the left hand side and right hand side, respectively.

A digital signature mechanism is called secure if it resists at least a **total break**, i.e., if it is infeasible for an adversary to figure out the signature key. Sometimes, however, one will not even tolerate weaker kinds of breaks like the following in order of decreasing severity [GoMR_88]:

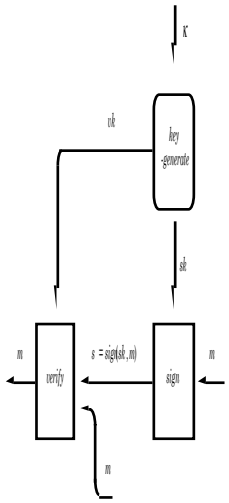

Fig. 5-1 Flow diagram for digital signature mechanisms

universal break: figure out an efficient signature algorithm which is functionally equivalent to *sign*,

selective forgery: forge a signature for some message chosen by the adversary,

existential forgery: forge a signature for any one message, even a random or non-sensical one.

> **Note:** The security of a well-designed digital signature mechanism does NOT rely upon the secrecy of its algorithms *sign* and *verify*, but solely upon the secrecy of its signature keys.

Digital signature mechanisms are asymmetric cryptographic mechanisms, i.e., it is infeasible to compute the matching signature key from a given verification key. Hence, signature keys are private keys whereas verification keys are public keys.

In order to achieve digital signature mechanism that produce signatures whose lengths do not depend on the lenghts of the messages signed, it is recommended to employ hash mechanisms (chapter 6). The operations of such signature mechanisms can be described as compositions of the hashing operation of a hash mechanism *H* and the appropriate operations of a **core signature mechanism** *G*. This approach is adopted because it profits by independent cryptologic research on the more elementary mechanisms. Furthermore, it allows to design signature mechanisms more appropriately to meet specific needs.

Fig. 5-2 gives the flow diagram of the key generating operation, the verifying and the signing operation of a digital signature mechanism which employs a hash mechanism.

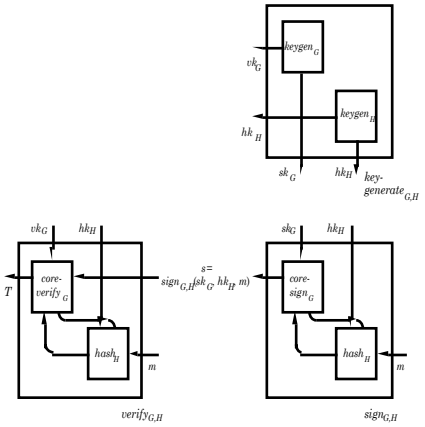

Fig. 5-2 Flow diagram for digital signature mechanisms that employ a hash mechanism

5.1 Proposed mechanisms

The following proposals provide the desired optimum with respect to security: a provably secure digital signature mechanism. Furthermore, an alternative mechanism is provided which is not known to be equally secure, but is more popular. Chapters 5.1.1 and 5.1.2 propose two signature mechanisms. Fig. 5-3 suggests which hash mechanisms (chapter 6) should be used for which core signature mechanism.

core signature mechanism *G*	**hash mechanisms *H***
GMR	pDAM
RSA	pDAM, wDES, wIDEA

Fig. 5-3 Proposal how to combine core signature mechanisms with hash mechanisms

The decision which hash mechanism to employ for RSA signatures depends on the overall security policy and on the performance requirements. In general, it is recommended to utilize some hash mechanism whose security relies on at least no other assumptions than the security of RSA does. This, e.g., applies to the hash mechanism by IVAN DAMGÅRD.

However, if large messages need to be signed and security is not the only issue one of the faster hash mechanisms based on DES or IDEA can be used.

5.1.1 GMR

In 1984, SHAFI GOLDWASSER, SILVIO MICALI, and RONALD RIVEST first published a digital signature mechanism provably secure against the strongest possible attack as far as one of the most thoroughly investigated complexity theoretic assumptions holds true (see chapter 5.3). After the names of its discoverers, the original mechanism has been referred to as **GMR**. It has subsequently been improved and demonstrated to be approximately as efficient as RSA (chapter 5.1.3). See the original work [Damg_88, GoMR_88, FoPf_91].

Note: Implementations of GMR require a multiple precision integer arithmetic (MPIA) in order to process numbers of 100 up to 400 decimal digits depending on the security parameter κ chosen.

Definition: GMR core signature

Producing and verifying signatures and generating keys is defined by the algorithms $sign_{GMR,pDAM}$, and $verify_{GMR,pDAM}$, $keygenerate_{GMR,pDAM}$, respectively. Two auxiliary functions f: $\{0,1\}^{\beta} \times \mathbb{N} \rightarrow \mathbb{N}$, $lp2$: $\mathbb{N}_0 \rightarrow \mathbb{N}_0$ and a probabilistic algorithm $d(\bullet)$ are needed. They are explained below.

Signing: The signing operation

$$s := sign_{GMR,pDAM}(sk, hk, m),$$

takes three inputs, the private signing key sk, the hashing key hk for the Damgård hash mechanism pDAM (cf. chapter 6.1.7), and the message m. It produces the digital signature $s = (nr', S)$, a number $nr' \in \{0, .., \Theta-1\}$, and a $(\theta+1)\times 2$ matrix of integers. The signing key $sk = (p, q, nr, M)$ consists of two primes p and q, a counter $nr \in \{0, .., \Theta-1\}$ of signatures produced by sk so far, and a $(\theta+1)\times 3$ matrix M of integers. The signing algorithm is as follows:

begin {of $sign_{GMR,pDAM}$}
  if $(2^{\theta} < nr)$ then
    abort operation since no more signatures can be produced from this sk.
  $M_{\theta+1,2} \leftarrow d(p \bullet q)$; { leaf signature }
  $M_{\theta+1,0} := hash_{pDAM}(hk, m, M_{\theta+1,2})$; { leaf reference }
  $L := lp2(nr)$
  For $l := \theta$ downto $L + 1$ do begin
    $M_{l,1} \leftarrow d(p \bullet q)$; { right child }
    $M_{l,2} \leftarrow d(p \bullet q)$; { node signature }
    $M_{l,0} := f(p \bullet q, M_{l+1,0} | M_{l+1,1}, M_{l,2})$; { node reference }
  end; { of For..}
  $(M_{L,0}, M_{L,1}) := (M_{L,1}, M_{L,0})$; { exchange entries }
    If $(L = \theta)$ Then { leaf signature }
    $M_{L,2} := f^{-1}(p, q, M_{L+1,0}, M_{L,0})$;
    Else { node signature }
    $M_{L,2} := f^{-1}(p, q, M_{L+1,0} | M_{L+1,1}, M_{L,0})$;
  $sign_{GMR,pDAM} := (nr, S)$, where $S_{i,0} = M_{i,1}$, $S_{i,1} = M_{i,2}$) for all $i \in \{0, .., \theta + 1\}$

$nr := nr + 1$;
end; {of $sign_{GMR,pDAM}$}

Verifying: The verifying operation

$$T := verify_{GMR,pDAM}(vk, hk, m, s)$$

takes four inputs, the public verification key vk, the hashing key hk for the Damgård hash mechanism pDAM (cf. chapter 6.1.7), the signed message m, and the signature s. It outputs T = TRUE if and only if s is a correct signature for m. The verification key $vk = (n, pr)$ consists of a modulus n, and a public reference pr. The signature $s = (nr, S)$ consists of a number $nr \in \{0, .., \Theta-1\}$, and a $(\theta+1)\times 2$ matrix S of integers. The verifying algorithm is as follows:

begin {of $verify_{GMR,pDAM}$} { declare a as an integer variable}
  if $(2^\theta < nr)$ then
    abort operation since no more signatures can be produced from this sk.
  $a := hash_{pDAM}(hk, m, S_{\theta+1,1})$; { leaf reference }
  $a := f(n, a, S_{\theta,1})$; { leaf node reference }
  For $l := \theta-1$ downto 0 do begin
    If odd($nr, \theta - l - 1$) then
    $a := f(n, S_{l+1,0}|a, S_{l,1})$ { take left child from signature}
    Else
    $a := f(n, a|S_{l+1,0}, S_{l,1})$ { take right child from signature}
  end; { of For..}
  $verify_{GMR,pDAM} := (a \stackrel{?}{=} pr)$
  $nr := nr + 1$;
end; {of $verify_{GMR,pDAM}$}

Key generation: The key generating operation

$$(sk, vk) \leftarrow keygen_{GMR,pDAM}(\kappa, \theta)$$

takes two inputs: the security parameter κ, and the capacity parameter θ. It computes a secret signing key $sk = (p, q, hk, \theta, nr, M)$ and a public verification key $vk = (n, hk, \theta)$, such that

- $(p, q, n) :\in_R T_{\kappa,10,20,R38,R78}$, where
  $R38 = \{p \in \mathbb{N} | p \equiv 3 \pmod 8\}$ and $R78 = \{p \in \mathbb{N} | p \equiv 7 \pmod 8\}$,
- $nr := 0$,
- $M_{i,j} := 0$ for all $i \in \{0, .., \theta + 1\}$, $j \in \{0, .., 2\}$,
- $hk \leftarrow keygen_{pDAM}(\kappa-1)$.

The knowledge of the prime factors p and q enables the signer to efficiently compute f^{-1} (and f) whereas knowledge of $n = p \bullet q$ enables a verifyer to only compute f efficiently. The counter nr is global to its corresponding signing key sk. It indicates how many signatures have been produced by sk so far. The matrix M stores those parts of previously produced signatures the signer can use for subsequent signatures.

The **function *odd*(*x, i*)** takes two non-negative integers with $i \in \{0, .., k\}$, $k = len(x)-1$, and returns TRUE if and only if the i-th bit x_i of the binary representation of $x = x_k x_{k-1}..x_0$ is 1.

The **function *lp2*(*x*)** takes a non-negative integer x and returns the "inverted" index of the least significant 1 in the binary representation of $x = x_k x_{k-1}..x_0$. A possible implementation is:

```
begin {of lp2}
    k := 0;
    while (k < len(nr)) and (not odd(x, k)) do k := k + 1;
    lp2 := len(nr) – k;
end; {of lp2}
```

The **function *f*(*n, m, x*)** takes an integer n, the product of two primes, an arbitrarily long binary string m and an integer $x \in \mathbb{ZZ}_n$, and returns a hash value according to the following algorithm:

```
begin {of f}                                   { declare a as an integer variable}
    a := 4^m x^(2^len(m)) (mod n);
    if (a < n/2) then f := a else f := n – a;
end; {of f}
```

The **function f^{-1}(*fpri, m, y*)** is the inverse of $f(fpub, m, \bullet)$, i.e., $f^{-1}(fpri, m, f(fpub, m, x)) = x$, for every corresponding pair (*fpri*, *fpub*) of keys.

```
begin {of f^-1}                                { declare a as an integer
    variable}
    ep := ((p+1)/4)^len(m) (mod p–1)
    eq := ((q+1)/4)^len(m) (mod q–1)
    a := cra(4^(–ep•rev(m)) y^ep (mod p), 4^(–eq•rev(m)) y^eq (mod q));
    if (a < n/2) then f^-1 := a else f^-1 := n – a;
end; {of f^-1}
```

The **algorithm *d*(*n*)** chooses a random element from $\mathbb{ZZ}_n$, such that $d < n/2$ and the Jacobi Symbol $(d/n) = +1$.

```
begin {of d}                                   { declare a as an integer variable}
    a :∈_P ZZ_n ;
    a := a^2 (mod n);
    if (a < n/2) then d := a else d := n – a;
end; {of d}
```

5.1.2 RSA

In 1978, RONALD RIVEST, ADI SHAMIR, and LEONHARD ADELMAN published the first digital signature mechanism, but could not base its proof of security upon some thoroughly investigated complexity theoretic assumption (see chapter 5.3). According to the names of its inventors, the original mechanism is called **RSA**. The mechanism has become quite popular and implementations of it are widely available nowadays. The original mechanism has subsequently been improved by [QuCo_82, Denn_84, Damg_88].

Note: Implementations of RSA require a multiple precision integer arithmetic (MPIA) in order to process numbers of 100 up to 400 decimal digits depending on the security parameter κ chosen.

Definition: RSA core signature

Producing and verifying signatures and generating keys is defined by the algorithms $sign_{\mathrm{RSA},H}$, and $verify_{\mathrm{RSA},H}$, $keygenerate_{\mathrm{RSA},H}$, respectively.

Signing: The signing operation

$$s := sign_{\mathrm{RSA},H}(sk, m),$$

takes two inputs, the private signing key sk, and the message m. It produces the digital signature s, an integer. The signing key $sk = (p, q, d)$ consists of two primes p and q, and a private exponent d. The signing algorithm is as follows:

begin {of $sign_{\mathrm{RSA},H}$} { declare a as an integer variable}
  $a := hash_H(m)$;
  $sign_{\mathrm{RSA},H} := cra(a^d \pmod p, a^d \pmod q)$;
end; {of $sign_{\mathrm{RSA},H}$}

Verifying: The verifying operation

$$T := verify_{\mathrm{RSA},H}(vk, m, s)$$

takes three inputs, the public verification key vk, the signed message m, and the signature s. It outputs T = TRUE if and only if s is a correct signature for m. The verification key $vk = (n)$ only consists of an integer modulus n. The signature s is an integer. The verifying algorithm is as follows:

begin {of $verify_{\mathrm{RSA},H}$} { declare a as an integer variable}
  $a := hash_H(m)$;
  $verify_{\mathrm{RSA},H} := (s^e \pmod n \stackrel{?}{=} m)$;
end; {of $verify_{\mathrm{RSA},H}$}

RSA keys are generated as follows:

$$(m, (p, q, d)) \leftarrow keygen_{\mathrm{RSA}}(\kappa)$$

begin {of $keygen_{\mathrm{RSA}}$}
  $(p, q, m) :\in_{\mathrm{P}} T_{\kappa,10,20,RSAPrimes,RSAPrimes}$,, where
    $RSAPrimes = \{p \in \mathbb{P}_{hard} | p' \in \mathbb{P}_{hard}$ for at least one factor p' of $p-1\}$,
  $e := 3$;
  $d := e^{-1} \pmod{(p-1)(q-1)}$;
end; {of $keygen_{\mathrm{RSA}}$}

Note: Implementations of RSA require a multiple precision integer arithmetic (MPIA) in order to process numbers of 100 up to 400 decimal digits depending on the security needed.

5.2 Choice of security parameter(s)

There is one primary security parameter κ relevant for any core signature mechanism G. κ determines the cardinality of the domain $SKEY_G$ of signature keys. Some mechanisms provide additional or optional parameters. Fig. 5-4 provides a range of proposed values for each core signature mechanism. Before picking some parameter chapter 5.4 should be consulted.

G	κ [bit]	b [#sign's]
GMR	{512, .., 1024}	> 0
RSA	{512, .., 1024}	–

Fig. 5-4 Security parameters proposed for the core signature mechanisms

5.3 Adversary models

Attacks are called **active** if the adversary is able to use the victim as an oracle by asking him questions and receiving his answers (typically without the victim noticing to be under attack). Otherwise, they are called **passive**. Passive attacks against a digital signature mechanism can be classified into

key-only attacks: the adversary only knows the verification key (public key) of the victim.

known-message attack: the adversary is additionally aware of some pairs (m_1, s_1), (m_2, s_2),… of corresponding messages and corresponding signatures.

Furthermore, active attacks are classified into:

directed-chosen-message attack: the adversary performs a known-message attack except that he himself chose the messages m_1, m_2… before.

adaptive-chosen-message attack: the adversary performs a directed-chosen-message attack, except that he did not have to choose all the messages m_1, m_2… in advance, but rather could have chosen them one after the other depending on the signatures he obtained before.

5.4 Cryptanalysis

The fundamental results are that digital signature mechanisms unconditionally secure for signers and verifiers are inefficient [ChRo_91]. However, there are digital signature mechanisms provably unconditionally secure for signers and computationally secure for verifiers and vice versa. The former are referred to as **classical digital signature** mechanisms whereas the latter are called **fail-stop digital signature** mechanisms. For both kinds of mechanisms there are efficient implementations [GoMR_88, HePe_93]. Either of their proofs of security only relies on one of the classical complexity theoretic assumptions: factorization of integers or discrete logarithm. More efficient classical digital signatures have been proposed whose security relies on some ad-hoc assumptions, e.g., DSS.

5.4.1 GMR

The security of GMR against existential forgery is proved to be equivalent to the classical complexity theoretic assumption of integer factorization. This holds even under the most serious attack (i.e. adaptive-chosen-message).

5.4.2 RSA

The security of core RSA against a total break has not been proven to be equivalent to one of the classical complexity theoretic assumption, not even if adversaries are restricted to passive attacks. Hence, the proof of security must be said to rest on an ad-hoc assumption, the RSA assumption. Hence, a universal forgery might also be possible. However, since the publication of RSA in 1978 no indication has come up that the RSA assumption might be wrong. Some passive attacks against RSA have been proposed in the literature [SiNo_77, Herl_78, Berk_82]. But none of them succeeds as [Rive_78, Rive_79, Berk_82] show. So far, no other attack has come up that is more likely to totally break core RSA than factorizing the modulus.

Core RSA is vulnerable to a selective forgery. A selective-chosen-message attack can make use of the homomorphism property of core RSA [Denn_84]. This attack does not apply if a hash mechanism is first applied to the message (chapter 5.1.1). However, the overall security of the RSA digital signature mechanism relies on the same ad-hoc assumption as core RSA.

Small exponents of RSA give rise to various passive attacks [Wien1_90] which can be avoided by choosing it as proposed.

5.4.3 How long does it take to forge a signature?

For the digital signatures proposed by chapter 5.1.1 no attacks are known that yield an existential, selected, or universal forgery in less time than a total break would cost. For a total break no better attack is known than factorization of the modulus of the core signature mechanism. (The key of the hash mechanism is publicly known.) The expected time to factor a modulus of given length can be taken from chapter 3.4.5.

5.4.4 Valuation of the proposals

There are other digital signature mechanisms like the one suggested by TAHER ELGAMAL [ElGa_85] or its variants [Schn_91, DSS1_91]. DSS is not proposed because it is at most as secure as core RSA composed with a hash mechanism based on DES or IDEA, and the first proposal (for DSS) fixes the key size to $k = 512$ bit. This, in particular, seems to be too short to be secure [Rive4_91] and additionally prohibits to adapt the key size to different security requirements. Recent proposals let the key size vary up to 1024 bit. This would prevent state of the art attacks in 1992, but will be too small to achieve integrity over a period of say 25 years [Rive4_91].

5.5 Implementations and their performance

Performances of signature mechanisms should be compared carefully. The signing and verifying operation of the RSA signature mechanism (E = DAM, G = RSA) and of the RSA core signature mechanism have asymptotically the same complexity as GMR. However, for short messages they differ significantly. This is demonstrated by a comparison of the

number $mm(l)$ of modular multiplications [23] of the respective algorithms. The complexity of modular multiplication is the dominating measure for at least software implementations.

For fixed modulus size κ = 512 bit, Fig. 5-5 compares the number of modular multiplications for RSA with and without a provably collision free hash function and for GMR. Modular multiplications are done with respect to two sizes of modulus, κ bit and (κ div 2) bit. This comparison is based on a modular multiplication algorithm that has quadratic complexity in the size of the modulus. Hence, the total number $mm(\kappa \text{ div } 2)$ to *sign*/*verify* is determined by using the equality

$$mm(2 \times (\kappa \text{ div } 2)) = 4 \times mm(\kappa \text{ div } 2)\ [24].$$

len denotes the length of a message in bit, *b* denotes a logarithmic bound for the maximum number $B = 2^b$ of signatures that can be generated from a single secret key as introduced in chapter 5.1.2.

$mm(\kappa$ **div 2)**	***sign***	***verify***
RSA, DAM	**Error!** • *len* + 1.5	**Error!** • *len*
core RSA	**Error!** • *len* + 502.5	**Error!** • *len* + 1940
GMR	**Error!** + 5034.7	4 • *len* + 512 (2*b*+1)

Fig. 5-5 Number of modular multiplications for RSA and GMR

This means that, for example, RSA performances (without a hash mechanism) may be almost linearly scaled for different lengths of messages whereas this does not hold for GMR performances.

To get the results as comparable as possible, the moduli and the signatures of both mechanisms should be fixed roughly to equal size. For a bound $B = 2^{10} = 1024$ a GMR signature is 11264 bit long. Thus the RSA signature was chosen to be 8192 bit = 1 Kbyte long which results in a message length *len* = 8192 bit. (This length is also taken in the literature sometimes). Hence all following performances refer to a modulus length of 512 bit, message length *len* = 8192 bit, signature bound (for GMR) B = 1024.

So far, only mechanism parameters were fixed. Hardware- and software performances depend on many other parameters. Hardware products usually give some of them (see the tables below) and keep others secret. Software performances depend heavily on the host machine used for the benchmark etc. Some of the details are found in the following chapters.

5.5.1 RSA hardware implementations

Six of the fastest RSA chips are given in the following table. Detailed descriptions about RSA hardware was sometimes not available; for example, if the chips are easy to use for digital signatures, if they have implemented any hash mechanism or if they make use of the **chinese remainder algorithm** (CRA) while signing. Mostly, performance of hardware

[23] *l* denotes the size of the underlying modulus in bit

[24] For a detailed description of the underlying mulitple precision arithmetic see [Fox_91].

is given by the deciphering bitrate or by the time needed to sign an RSA block. If the bitrate was given it was converted into the (average) time needed to sign one RSA block. The following table gives the state of the art of RSA hardware, but comparing the performances has to take into account at least three parameters:

i) clockrate, ii) chip size, iii) size of modulus .

(The number of bits processed per chip indicates, how many chips have to be cascaded in order to process blocks of larger size.)

	avail-ability	**technology [µm]**	**chip size [mm^2]**	**#bits / chip**	**clock [MHz]**	**sign one block time [s] bitrate [Kbit/s] (size of modulus)**	
Cryptech [Bric_90]	1988	Gate Array	?	120	14	0.482 (512)	1.062
[Sedl_88]	1989	5	4.8 × 5.0	780 [25]	?	0.042 (780)	18.571
VICTOR [OrSA_91]	1990	2	10.0×10.0	512	20	0.0845 (512)	6.059
[VVDJ_90]	1990	2 (CMOS)	9.3 × 8.7	1024	25	1.024 (1024)	1.000
CORSAIR [WaQu_91]	1991	1.2 (CMOS)	1.7 × 1.7	512	6	1.5 (512) without CRA	0.341
Philips DX-Card	1992	?	1.7 × 1.7	512	8	0.4 (512)	1.280

Fig. 5-6 Performance of RSA-Hardware

5.5.2 RSA software implementations

Performances were calculated while signing/verifying 16 blocks of message (*len* = 16 × 512 bit = 8192 bit = 1 Kbit) at a modulus size of 512 bit. No hash mechanism is used. Signing is done by using the chinese remainder algorithm.

The verifying was done using a large public exponent as it was suggested for RSA originally. Choosing the public exponent smaller, say of size 15…20 bit would increase the verifying rate by a factor of 10 or more (assumed modulus size is 512 bit). This applies to both hardware and software implementations. No attacks were found whose probability of success increases significantly if applied to a verification exponent of 3.

	avail-ability	**supported processor**	**precom-putation [Kbyte]**	**reference machine**	**sign one block verify one block time [s] bitrate [Kbit] (size of modulus)**	
RSA	1993	680x0 x ∈ {0…4}	0	Apple [26] Quadra950	2.05 (512) 6.65 (512)	3.948 1.232

[25] This is achieved under the reasonable restriction that the prime factors of the modulus are of sizes 340 bit and 440 bit respectively.

[26] Apple Macintosh Quadra 950 (MC68040, 30 MHz, 80 ns RAM)

RSA [Fox_91]	1991	80x86 x ∈ {1…4}	0	386 Clone [27]	9.095 (512) 30.74 (512)	0.901 0.266
Rescrypt [Resc_91]	1991	80x86 x ∈ {1…4}	?	386 Clone [28]	5.28 (512) ≈ 18 (512)	1.552 0.455

Fig. 5-7 Performance of RSA-Software

5.5.3 GMR hardware Implementations

So far, no actual hardware implementation of the GMR signature mechanism is known to the author.

5.5.4 GMR software Implementations

Times are given to sign 8,192 bit ≈ 1 Kbyte of message/to verify the signature of 8,192 bit of message at a modulus size of 512 bit. The upper bound *B* of possible signatures per key was chosen to be *B* = 1024 for comparability reasons.

	avail-ability	**supported processor**	**precom-putation [Kbyte]**	**reference machine**	**sign one block verify one block time [s] bitrate [Kbit] (size of modulus)**	
GMR [FoPf_91]	1993	680x0 x ∈ {0…4}	0	Apple [29] Quadra950	1.6 (512) 8.26 (512)	5.120 0.990
GMR [Fox_91]	1991	80x86 x ∈ {1…4}	0	386 Clone [30]	3.905 (512) 61.08 (512)	2.098 0.134

Fig. 5-8 Performance of GMR-Software

5.6 Standardization

An international standard for digital signature mechanisms is currently being prepared. One first proposal is the digital signature standard [DSS_91]. So far there are de-facto standards developed by companies as RSA Data Security, Inc. A good overview is found in [Kali3_91].

6 HASH MECHANISMS

Hash mechanisms are an important cryptographic tool. Informally, the purpose of a hash mechanism *H* is to map some long binary sequence to some hash value of fixed length in such a way that it is hard to find another sequence which maps to the same hash value. In

---

27 Processor: Intel 80386, 33 MHz, 64 Kbyte cache board
28 Processor: Intel 80386, 33 MHz
29 Apple Macintosh IIfx (MC68030, 40 MHz, 32 Kbyte cache board, 80 ns RAM)
30 Processor: Intel 80386, 33 MHz, 64 Kbyte cache board

order to achieve this property, some hash mechanisms employ a cryptographic encipherment mechanism. Let M, HV be the respective domains of binary sequences and hash values. In general, a hash mechanism provides one characteristic functional operation (usually called a **hash function**)

$$hash\colon \{HKey \times HV \times\}\ M \rightarrow HV$$

Some hash mechanisms also take a hashing key from the set $HKey$ or an initial value from the set HV as an optional input. If the mechanism requires a hashing key to be input into the hash operation, it also provides an inderterministic key generating operation

$$hk \leftarrow keygenerate(\eta)$$

which on input a security parameter η outputs some hashing key.

In the literature hashing is sometimes called "compression" because it potentially shortens the input sequence. However, the term compression might be misleading because i) for input sequences shorter than the hash values, hashing in effect expands the input sequence, and ii) in coding theory the term compression is reserved to mappings which preserve the information of their input, e.g, by eliminating redundancy. However, hashing does not preserve the information of input, i.e., in general, the input cannot be obtained from its hash value in an information theoretic sense.

Two kinds of hash mechanisms are distinguished: A **keyed hash mechanism** is parametrized by either a private key (if the underlying encipherment mechanism E is symmetric) or a public key (if E is asymmetric). A **one-way hash mechanism** is not parametrized by a key. Hence, one-way hash mechanisms can be interpreted as special cases of keyed hash mechanisms, namely ones with a constant key.

Fig. 6-1 gives the flow diagram for hash mechanisms. Deterministic algorithms are indicated by square boxes whereas indeterministic algorithms are given by capped boxes.

Besides the above sketched modular design of hash mechanisms, there are some monolithically designed one-way hash mechanisms, e.g. MD4, MD5, SHS [Rive2_91, Rive_91, SHS_92]. The former two were broken by [BoBo_92, BoBo_94], [Bers_93], respectively. The latter has still faced too little cryptanalysis to be proposed by now.

It is proposed to employ **iterated hash mechanisms**. They are constructed by either a linear or a logarithmic iteration of some hash round mechanism [LaMa_93]. The hash round mechanism in its turn might employ an encipherment mechanism (chapter 3).

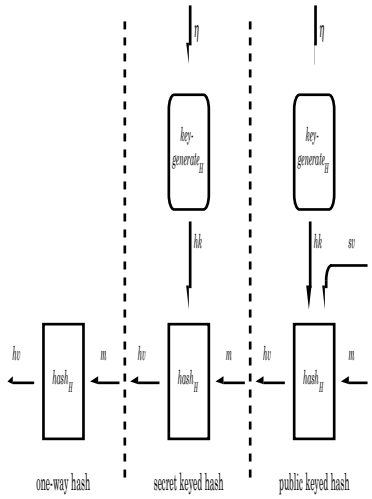

Fig. 6-1 Flow diagrams for hash mechanisms

The modular construction implies that **iterated keyed hash mechanisms** can only be obtained by **keyed hash round mechanisms**. Analogously, **iterated one-way hash mechanisms** are obtained by **one-way hash round mechanisms**. Fig. 6-2 summarizes which of the possible combinations are proposed and what they are called.

hash mechanism *H* (encipherment *E*)	**one-way**	**secret keyed**	**public keyed**
linear	wDES (DES)	sDES (DES), sIDEA (IDEA)	pDAM (DAM[31])
logarithmic	—	sCW1 (CW1)	—

Fig. 6-2 Proposal how to iterate round functions based on different encipherment mechanisms

Since the proposed encipherment mechanisms are block oriented binary sequences of arbitrary length first have to be padded before they can be fed to the proposed iterated hash mechanisms.

6.1 Proposed Mechanisms

Chapter 6.1.1 to 6.1.4 describe how one-way hash round mechanisms, secret keyed hash round mechanisms, and public keyed hash round mechanisms are iterated. Chapter 6.1.5 defines the sequence padding. Finally, four hash round mechanisms are proposed: rnd_{wDES} in chapter 6.1.6, rnd_{sDES}, rnd_{sIDEA} in chapter 6.1.7, and rnd_{pDAM} in chapter 6.1.8.

[31] since DAM is not proposed as an encipherment mechanism in its own it is described in the context of hash mechanisms rather than in chapter 3.1.

6.1.1 Iterated one-way hash mechanisms

Let H be a hash mechanism based on a block encipherment mechanism E (Fig. 6-2) with PB_E, CB_E being equal domains of plaintext blocks and ciphertext blocks, i.e., $B := PB_E = CB_E = \{0,1\}^\beta$,
$S = B^+$ be the set of all binary sequences of length a multiple of β,
$EKey_E$ be the set of encipherment keys of mechanism E, and let

$$pad_H: \{0,1\}^+ \to S,\ aux_H: B \to B,\ rnd_H: S \times B \to B$$

be a padding function, an auxiliary function, and a one-way hash round mechanism each specific for the selected E. Then, the **iterated one-way hash mechanism**

$$hash_H: S \to B \quad \text{for } H \in \{ \text{ wDES } \}$$

is constructed by the linear iteration

$$hash_H(pad_H(s)) := aux_H(b_n),$$

$$\text{for all } i \in \{1, .., n\} \text{ let } b_i := rnd_H(s_i, b_{i-1}), \text{ and } b_0 := \underbrace{00\cdots01}_{\beta-1}$$

where $pad_E(s) = s_1|s_2|\ldots|s_n \in S$, $b_i \in B$. The padding function pad_H is described in chapter 6.1.5, the auxiliary functions aux_H and the one-way hash round mechanisms rnd_H are proposed in chapter 6.1.6.

6.1.2 Iterated secret keyed hash mechanisms (linear construction)

Let H be a hash mechanism based on a block encipherment mechanism E (Fig. 6-2) with PB_E, CB_E being equal domains of plaintext blocks and ciphertext blocks, i.e., $B := PB_E = CB_E = \{0,1\}^\beta$,
$S = B^+$ be the set of all binary sequences of length a multiple of β,
$EKey_E$ be the set of encipherment keys of mechanism E, and let

$$pad_H: \{0,1\}^+ \to S,\ aux_H: EKey_E \times B \to B,\ rnd_H: EKey_E \times S \times B \to B$$

be a padding function, an auxiliary function, and a keyed hash round mechanism each specific for the selected mechanism H. Then, the **iterated keyed hash mechanism**

$$hash_H: HKey_H \times S \to B \quad \text{for } H \in \{ \text{ sDES, sIDEA } \}$$

is constructed by the linear iteration

$$hash_H(hk, pad_H(s)) := aux_H(hk, b_n),$$

$$\text{for all } i \in \{1, .., n\} \text{ let } b_i := rnd_H(hk, s_i, b_{i-1}), \text{ and } b_0 := iv_H$$

where $pad_H(s) = s_1|s_2|\ldots|s_n \in S$, $b_i \in B$. The padding function pad_H is described in chapter 6.1.5.[32] The auxiliary functions aux_H and hash round mechanisms rnd_H are described in chapters 6.1.7. The value of the variable iv_H depends on the hash round mechanism H actually chosen (see chapter 6.1.6).

---

[32] It is clear from this construction, that the length η of the hash values equals the length β of the blocks of ciphertext of the corresponding block encipherment mechanism E ($HV_H = CB_E$). E.g., employing E = DES yields 64 bit hash values.

The generation of keys is defined as

$$keygenerate_H = keygen_E,$$

6.1.3 Iterated secret keyed hash mechanisms (logarithmic construction)

A highly secure hash mechanism is found in [CaWe_79, WeCa_81]. Its iteration is somewhat more complicated than the linear one (chapter 6.1.2), but it is provably secure in an information theoretic sense.

6.1.4 Iterated public keyed hash mechanisms

Let H be a hash mechanism based on a block encipherment mechanism E (Fig. 6-2) with PB_E, CB_E being equal domains of plaintext blocks and ciphertext blocks, i.e., $B := PB_E = CB_E = \{0,1\}^\beta$,
$S = B^+$ be the set of all binary sequences of length a multiple of β,
$EKey_E$ be the set of encipherment keys of mechanism E, and let

$$pad_H: \{0,1\}^+ \to S,\ aux_H: EKey_E \times B \to B,\ rnd_H: EKey_E \times S \times B \to B$$

be a padding function, an auxiliary function, and a keyed hash round mechanism each specific for the selected mechanism H. Then, the **iterated public hash mechanism**

$$hash_H: HKey_H \times S \times B \to B, \quad \text{for } H \in \{\text{ pDAM }\}$$

is constructed by the linear iteration

$$hash_H(hk, pad_H(s), iv) := aux_H(hk, b_n),$$

$$\text{for all } i \in \{1, .., n\} \text{ let } b_i := rnd_H(hk, s_i, b_{i-1}), \text{ and } b_0 := iv$$

where $pad_H(s) = s_1|s_2|\ldots|s_n \in S$, $b_i \in B$. The padding function pad_H is described in chapter 6.1.5. The auxiliary functions aux_H and hash round mechanisms rnd_H are described in chapters 6.1.8.

The generation of keys is defined as

$$keygenerate_H = keygen_E,$$

6.1.5 Padding of binary sequences

The following padding functions are proposed:

$$pad_H := pad(64, \bullet) \text{ for } H \in \{\text{wDES, sDES, sIDEA, sCW1}\},$$

$$pad_H = sfx \circ pad(8, \bullet) \text{ for } H \in \{\text{pDAM}\},$$

The suffix-free encoding sfx (see below) is to be applied, since otherwise the hash round mechanisms rnd_{pDAM} (chapter 6.1.7) could not be proved to be collision-free (chapter 6.4). The padding function pad is defined in chapter 3.1.2.

Definition: On binary sequences of length a multiple of β the code $\boldsymbol{sfx}$ is defined as follows:

$$sfx: \{\{0,1\}^\beta\}^+ \to \{\{0,1\}^\beta\}^+$$

$$sfx(s) := s|\underbrace{0}_{e_{n-1}}|l_{n-1}|\underbrace{1}_{e_{n-2}}|l_{n-2}|\ldots|\underbrace{1}_{e_0}|l_0, \text{ where}$$

$n = 1 + len(s) \text{ div } 2^{\beta-1}$ denotes the number of appended blocks,

$l_{n-1}|l_{n-2}|\ldots|l_0 = len(s)$, $l_{n-1}, l_{n-2}, .., l_0 \in \{0,1\}^{\beta\text{-}1}$, and

the extension bits $e_{n-2}, .., e_0$ are 1, whereas $e_{n-1} = 0$.

Informally, this code does the following. It writes the length of a sequence behind the sequence in a field of length β. If the binary representation of the length of s does not fit into this field further fields are appended. Hence, each field is leaded by an extension bit which indicates if another field follows "to the left".

Definition: A code $f: A \rightarrow B$ is called **suffix-free** iff

1) f is bijective and
2) no element of B is the suffix of another element of B.

Lemma 6.1 The code sfx is suffix-free.

Proof:

Assume there exist two different binary sequences s, s' such that

$sfx(s') = s'|e'_{n'-1}|l'_{n'-1}|e'_{n'-2}|l'_{n'-2}|\ldots|e'_0|l'_0$ is a suffix of

$sfx(s) = s|e_{n-1}|l_{n-1}|e_{n-2}|l_{n-2}|\ldots|e_0|l_0$.

Now, let $v = min(n, n')$. Then for each $i \in \{0, 1, .., v\text{-}1\}$ one has $e'_i = e_i$ and $l'_i = l_i$ because all these components have respective fixed lengths of 1 bit or λ-1 bit. By definition of v it follows that $e'_{v-1} = 0$ and/or $e_{v-1} = 0$ and hence $e'_{v-1} = e_{v-1} = 0$. Thus, v is the smallest such index and by definition of sfx it follows that $n = n' = v$.

Hence, $len(s) = len(s')$ and the assumption that $sfx(s')$ is a suffix of $sfx(s)$) assure that $s' = s$ which is a contradiction to the precondition. ❑

6.1.6 One-way hash round mechanisms

A one-way hash round mechanism is proposed according to [PrGV_93, ISO10118-2].

Definition: Let $s, b \in B$ be the set of blocks of plaintext of DES, then

$$\boldsymbol{rnd}_{\mathbf{wDES}}(s, b) := s \oplus enc_{\text{DES}}(b, s)$$

$$\boldsymbol{aux}_{\mathbf{wDES}}(b) := b$$

6.1.7 Secret keyed hash round mechanisms

Two high speed keyed hash round mechanisms are proposed according to [RIPE2_93].

Definition: Let $ek \in EKey_{\text{DES}}$, $s, b \in B$ the set of blocks of plaintext of DES, then

$$\boldsymbol{rnd}_{\mathbf{sDES}}(ek, s, b) := s \oplus enc_{\text{DES}}(ek, s \oplus b)$$

$$\boldsymbol{aux}_{\mathbf{sDES}}(ek, b) := enc_{\text{DES}}(ek \oplus 0\text{x}\underbrace{\text{F0 F0} \ldots \text{F0}}_{8}, b)$$

$$\boldsymbol{iv}_{\mathbf{sDES}} := 0\text{x}\underbrace{00\ 00 \cdots 00}_{8}$$

Definition: Let $ek \in EKey_{\text{IDEA}}$, $s, b \in B$ the set of blocks of plaintext of IDEA, then

$$\boldsymbol{rnd}_{\mathbf{sIDEA}}(ek, s, b) := s \oplus enc_{\mathrm{IDEA}}(ek, s \oplus b)$$

$$\boldsymbol{aux}_{\mathbf{sIDEA}}(ek, b) := enc_{\mathrm{IDEA}}(ek \oplus 0\mathrm{x}\underbrace{\mathrm{F0\ F0\ \ldots\ F0}}_{16}, b)$$

$$\boldsymbol{iv}_{\mathbf{sIDEA}} := 0\mathrm{x}\underbrace{\mathrm{00\ 00\ \cdots\ 00}}_{8}$$

6.1.8 Public keyed hash round mechanism

One provably secure keyed hash round mechanism is proposed according to [Damg_88].

Definition: Let $ek \in EKEY_{\mathrm{DAM}}$, $s, b \in B$ be the set of blocks of "plaintext" of DAM, then

$$\boldsymbol{rnd}_{\mathbf{pDAM}}(ek, s, b) := a_s\, b^2 \pmod m$$

$$\boldsymbol{aux}_{\mathbf{pDAM}}(ek, b) := b$$

$$(ek, dk) \leftarrow \boldsymbol{keygen}_{\mathbf{DAM}}(\kappa)$$

where the input κ is a security parameter. The operation $keygen_{\mathrm{DAM}}$ outputs an enciphering key ek and a deciphering key dk. The latter is needed for special purposes only.

begin {of $keygen_{\mathrm{DAM}}$} { declare a, a' as integer variables}

$\beta := 8$; $\gamma := 2^{\beta} = 256$; { For the selection of β also refer to chapter 6.2, 6.5}

$(p, q, n) :\in_P T_{\kappa,10,20,P34,P34}$, where $P34 = \{p \in \mathbb{N} \mid p \equiv 3 \pmod 4\}$.

$a' := (a_0', a_1', .., a_{\gamma-1}')$, where the a_i' are independently chosen from $\mathbb{Z}_n^*$.

$a := (a_0, a_1, .., a_{\gamma-1})$, where $a_i := a_i'^2 \pmod n$.

$ek := (n, a)$; $dk := (p, q, a)$

end; {of $keygen_{\mathrm{DAM}}$}

6.2 Choice of security parameters

For $H \in$ {wDES, sDES, sIDEA, sCW1} $hash_{\boldsymbol{H}}$ provides no parameters. For $H \in$ {pDAM} it is proposed to select a security parameter $\kappa \in$ {512, .., 768} and a block size $\beta \in$ {1, .., 8} bit depending on the amount of memory available (cf. chapter 6.5).

6.3 Models of adversary control

Generally, hash mechanisms take three inputs: a cryptographic key ek, a binary sequence s which is to be hashed and some initial value iv. Provided the key is not known to the adversary, he might start the following attacks depending on which parameter he is free to choose [LaMa_93].

Fix-start target: Given s and iv, find s' such that
$s' \neq s$, but $hash(ek, iv, s') = hash(ek, iv, s)$.

Free-start target: Given iv and s, find iv', s' such that
$(iv', s') \neq (iv', s)$, but $hash(ek, iv', s') = hash(ek, iv', s)$.

Fix-start collision: Given iv, find s and s' such that
$s' \neq s$, but $hash(ek, iv, s') = hash(ek, iv, s)$.

Free-start collision: Find iv, iv', s, s' such that
$(iv', s') \neq (iv, s)$, but $hash(ek, iv', s') = hash(ek, iv, s)$.

The following inclusions (Fig. 6-3) hold between these attacks. ($A \subset B$, $\subset_B$ both denote that every attack of *A* is also an attack of *B*.)

$$\begin{array}{ccc} \text{fix-start target} & \subset & \text{fix-start collision} \\ \subset & & \subset \\ \text{free-start target} & \subset & \text{free-start collision} \end{array}$$

Fig. 6-3 Relations of attacks against hash mechanisms

Iterated one-way hash mechanisms and secret keyed hash mechanisms fix their initial values iv_H and hence are only subject to fix-start attacks. Public keyed hash mechanisms provide the initial value as an additional input to the hash operation. Thus in general, they can be subject to free-start attacks, too. In practice, however, a free start attack can only be mounted if the application using the hash mechanism does not fix the initial value by itself. For example, the core signature mechanisms (chapter 4) do not do so.

6.4 Cryptanalysis

Since hash mechanisms give rise to more powerful attacks (free-start attacks) if their initial values may be chosen by an adversary only hash mechanisms with fixed initial values are proposed. It is an open question if suffix-free encoding is necessary to yield security against the weaker fix-start attacks. At least no successful fix-start attack is presently known. Hence, a suffix-free encoding is proposed only for $hash_{\text{DAM}}$ whose proof of security requires the suffix-free encoding.

It is a difficult task to design secure, but efficient hash mechanisms. An excellent overview over the history of broken hash mechanisms and a systematic synthesis of secure ones is given by [PrGV_93]. This applies particularly to the keyed hash mechanism proposed by [DaPr_84, DaPr_89] and the one-way hash mechanism standardized [ISO9797]. For more detail also consult [PrGV2_93]. However, there can be no proof of collision-freeness for hash mechanisms which are based on unproved encipherment mechanisms. Hence, new suggestions for efficient iterated hash mechanisms must be expected in the future. Under development are the construction of hash mechanisms whose hash values are longer than the block size of the underlying block encipherment algorithm [HLMW_93] and differential cryptanalysis of hash mechanisms [PrGV4_93].

The hash mechanism based on [Damg_88] is provably collision-free, i.e., secure against fix-start collision attacks, under the assumption that factoring integers from *T* (chapter 2) is computationally infeasible.

6.5 Performance

The hashing performance of $hash_H$, $H \in$ {pDAM, sDES, sIDEA} is approximately the encipherment performance of the respective encipherment mechanism. The performance of $hash_{\text{wDES}}$ additionally depends upon the time needed to convert a 64 bit DES key into its corresponding 16 subkeys.

The "encipherment" operation of DAM costs one modular multiplication and one modular squaring. Thus its performance depends upon the security parameter κ and also upon the block size β. On the one hand, doubling β doubles the hashing performance (bits hashed per second). On the other hand, the size of the hashing key *ek* is $\kappa\,(1+2^{\beta})$, which is

exponential in β. For example, security parameter κ = 512 and β = 8 bit yields a hashing rate of approximately 1 bit per time to compute 1.75 modular multiplications [Fox_91, ch. I.5] and a key size of 1.5 Kbit.

6.6 Standardization

Still under consideration. [ISO10118, SHS_92]. [ISO9797] is no longer state of the art.

7 COMBINED MECHANISMS

Any of the mechanisms proposed by chapter 3 to 6 can be combined in order to yield several properties for one input. Besides applying different mechanisms sequentially there are sometimes equivalent, but more efficient solutions. These are called combined mechanisms. Confidentiality and detectability of modification are an example for two cryptographic building blocks whose joint functionality may be achieved by a special mechanism (chapter 7.1).

7.1 Iterated encipherment mechanisms with detection of modification

In order to simultaneously achieve confidentiality and detection of modification it is proposed to operate the selected (symmetric or asymmetric) block encipherment mechanism in **plain cipher block chaining (PCBC)** mode instead of CBC mode (chapter 3.1.3). Also see [MeMa_82, pp. 69-71] who call this mode "block chaining using plaintext and ciphertext feedback". The security of PCBC is supposed to be that of CBC. The performance of PCBC is almost that of CBC; it differs only by a small additive constant.

The primary operations *encipher-d_E*, and *decipher-d_E* of an **iterated encipherment mechanism with detection of modification** are constructed as follows:

$$\textit{encipher-}d_E\text{: } EKey_E \times P \rightarrow C$$

$$\textit{encipher-}d_E(ek, p) := cb_1|cb_2|\ldots|cb_n|cb_{n+1},$$

where for all $i \in \{1, .., n+1\}$: $cb_i := enPCBC(ek, pb_i, pb_{i-1}, cb_{i-1})$, $cb_0 := 0$,
$pb_i \in PB_E$, $pb_1\|pb_2\|\ldots\|pb_n := enpad(\beta, redins_E(p))$, $pb_{n+1} := 0$, β the length of plaintext blocks of E according to *ek*.

$$\textit{decipher}_E\text{: } DKey_E \times C \rightarrow P \times \{\text{TRUE, FALSE}\}$$

$$\textit{decipher-}d_E(dk, c) := (redchk_E(depad(\beta, pb_1|pb_2|\ldots|pb_n)), ok),$$

where for all $i \in \{1, .., n+1\}$: $pb_i := dePCBC(dk, cb_i, cb_{i-1}, pb_{i-1})$, $pb_0 := 0$,
$cb_i \in CB_E$, $cb_1|cb_2|\ldots|cb_n := c$, and ok:= TRUE iff $pb_{n+1} = 0$, β the length of plaintext blocks of E according to *dk*.

The padding and depadding function (*enpad*, *depad*) are defined in chapter 3.1.2. The reduncancy predicate (*redins*, *redchk*) is defined in chapter 3.1.3. Plain cipher block chaining is defined below.

Definition: Let E be some block encipherment mechanism, β its respective block length.
Plain Cipher Block Chaining (PCBC) is defined as follows.

$$enPCBC: EKey_E \times PB_E \times PB_E \times CB_E \rightarrow CB_E,$$

$$enPCBC(ek, pb, pb', cb) := enc_E(ek, ((pb' + cb) \bmod 2^\beta) \oplus pb),$$

$$dePCBC: DKey_E \times CB_E \times CB_E \times PB_E \rightarrow PB_E$$

$$dePCBC(dk, cb, pb, pb') := ((pb' + cb') \bmod 2^\beta) \oplus dec_E(dk, cb)$$

Fig. 7-1 Flow diagram for the mode of operation according to chapter 3.1.3

8 SECURE TRANSFER OF HEALTH CARE DATA

SEISMED aims at a secure European medical network (**MedNet**) that is capable of interconnecting medical health care centers and possibly patient's households in a safe and secure way. Secure data transmission requires technical measures to obtain confidentiality, authentication, and non-repudiation which is commonly agreed to be achieved by cryptographic mechanisms.

Chapter 8.1 outlines the subject under consideration and summarizes the main assumptions about MedNet with respect to utilizing cryptographic mechanisms. Chapter 8.2 presents the asymmetric key management and certification concept of MedNet. Chapter 8.3 gives the details where the MedNet key management differs from the standard [ISO9594-8] and why. Chapter 8.4 covers non-repudiation. High speed encipherment finally requires MedNet users to establish symmetric keys which is dealt with in Chapter 8.5.

8.1 Scope

The first step towards a European MedNet will be the interconnection of existing health care information systems (HISs). Future developments in health care may suggest or require to also connect private households of patients to MedNet in order to provide certain medical services to patients at their homes. Hence, the network should be designed to be an open network in the sense that there will be a large varying group of participants using that network. This recommendation for key management should not restrict the selection of a network. But as an example one could think of the ISDN to be the physical basis for MedNet.

This chapter deals with the management (generation, distribution, updating, revocation) of keys needed for confidentiality, detection of modification, and proofs of origin within MedNet. For confidentiality this chapter provisionally assumes an encipherment mechanism, that

- is allowed to be used for encipherment of medical data transmitted over a public network in every EC-member state and

- is accepted by the patients, medical community, health insurances and the national health authorities of all EC-member states.

If such an encipherment mechanism turns out not to exist, one has to select one that meets the above criteria of most the EC-member states. Those who do not accept it, have to maintain one or more national links that decipher and possibly re-encipher all input from MedNet by some appropriate system and vice versa for all output to MedNet. Connection to MedNet and maintenance of such national security domains with a security policy different from that of MedNet is outside the scope of this chapter.

The group of participants will potentially comprise several thousands (health care centers) up to several million (households included) of participants. This is called the requirement $\boldsymbol{R_1}$ of the **size of the group of participants**. One consequence of R_1 is that exchanging a number of session keys quadratically increasing in the number of participants were impractical. Hence, it is recommended to utilize public-key encipherment mechanisms.

Sensitivity of personal medical data, European and national legislation demand the network to provide confidential, authentic, and non-repudable transmission of data. Secret means that only the parties involved within a certain medical treatment become aware of the data relevant to this treatment. This requirement $\boldsymbol{R_2}$ of potentially **confidential and/or authentic communication** excludes to have non-public keys managed by national, supranational or European Key Distribution Centers (KDCs). R_2 thus also excludes the use of identity-based cryptographic mechanisms as first proposed by [Sham_85].

The set U of potential participants of MedNet thus comprises: physicians at possibly different departments of health care environments (HCEs), general practitioners, and patients – all in possibly different districts in possibly different countries. MedNet, thus, has to reflect the hierarchical interdependencies of these users and their corresponding organizational entities. From the point of view of the ISO OSI model [ISO7498] protection has to be provided at least at three layers:

(1) R_2 requires to protect the personal communication between a patient and the physician actually responsible for him. This requires to encipher/decipher some or all data at the application layer. From the view of patient-physician relations their user processes are the endpoints of communication. In this sense R_2 requires **end-to-end encipherment**. This, however, does not keep other participants on the network (even outside the considered hospital) from learning something about the digital traffic between the communicating partners.

(2) Hence, protecting the (temporary) network addresses and thereby the identity of the communicating partners further requires encipherment at the lowest possible layer. If health care centers utilize public networks the lowest possible layer under their control will be the transport layer. From the view of patient-physician relations their health-care information systems are the links of communication. In this sense **link-by-link encipherment** has to be provided.

(3) The network itself should also provide an enciphering service on the physical or data-link layer in order to protect addressing information from wiretappers.

As physicians and patients are mobile users their desired keys have to be provided at the right terminal at the right time. A user should be encouraged to change his public key periodically (every 2-5 years) since due to his mobility he has to entrust even his secret keys

to many different machines. Furthermore, a user should be enabled to change a secret key on demand if he suspects that his actual key is corrupted. Health care information systems are comparatively static entities within MedNet and, thus, need not change their public keys frequently. The switching centers of the underlying public network may operate the link-by-link encipherment with fixed keys for the whole of their lifetime. Hence, end-to-end encipherment (1) puts the strongest requirements upon MedNet key-management.

The basis for the following recommendation is the Directory – Authentication framework [CCITT509, ISO9594-8] and a constructive critique [AnMi_90] of this proposed standard. The following recommendation satisfies the requirements (1) and should, thus, also satisfy the weaker requirements (2) and (3). In the following, only [ISO9594-8] is cited because it already avoids some weaknesses of the 1988 **CCITT509** recommendation. Nevertheless, it must be stressed that even [ISO9594-8] has not eliminated all weaknesses. All modifications and improvements to [CCITT509] and [ISO9594-8] are explicitly given by chapter 8.3.

8.2 Asymmetric key management of MedNet

The main goal of key-management is to provide message confidentiality and authentication to MedNet users. This is only possible by utilizing public-key encipherment and digital signature mechanisms. In contrast to the suggestions of [ISO9594-8]) each user must generate different keys to drive different mechanisms in order to avoid breaches of security.

8.2.1 Distinguished name service and key distribution

In order to associate a unique user to every name (not necessarily vice versa) a distinguished name service is provided by **Distinguished Name Centers** (DNCs) of MedNet. Every MedNet user can establish at least one pair of keys for encipherment (*ek*, *dk*) and for digital signing (*sk*, *vk*), respectively. To support users in publishing their public keys **Key Distribution Centers** (KDCs) are provided by MedNet. Their task is to deliver public keys to anyone who requests them[33]. If users wish to publish their public keys to several KDCs they may do so. The resulting redundancy of key storage prevents single KDCs from getting too powerful.

The canonical way to generate a user's secret and public key(s) were to encourage each user to generate them on his own. He would then deliver the public key(s) to his KDC(s) to register them. Depending on the application a user is to more or less identify himself before being allowed to register a public key.

If the security policy wants health care authorities to monitor communication via MedNet, these authorities must be provided with the user's secret keys. Hence, they could generate keys on their own and deliver them to users on demand or users could be enforced to come up with the corresponding secret key for every public key they want to register at a KDC.

A compromise between patient's rights on informational self determination and democratic control over MedNet communication could be established by storing secret keys by secret sharing mechanisms [Sham_79] at different sites (e.g. KDCs). A minimum number of such sites were necessary to recover a secret key from the union of their knowledge. Distributing secret key information over KDCs that are controlled by different

---

[33] KDCs might also provide the distinguished name service if this turns out to be more efficient.

democratic powers this way could fulfill patient's and health care authorities rights and needs.

8.2.2 Certification of public keys

Security of asymmetric cryptography basically rests upon the distribution of authentic keys. For this purpose MedNet provides a certification service, which is most effectively gained by a hierarchy of **Certification Authorities** (CA). Each such CA initially provides a certificate for the public keys of its immediate predecessor (parent CA) and its immediate successors (child CAs) or of user's public keys. The certificates mainly consist of a digital signature for the item to be certified. This standard approach is called **strong authentication** according to [ISO9594-8, Section 3]. To initialize the authentication tree each user u of MedNet is initially acquainted with one such certification authority CA_u e.g. one located near to his home or health care center etc. This acquaintance is established by u and CA_u exchanging their public keys in an authentic way, e.g. by u visiting CA_u. By this initial authentication user u is enabled to recursively verify the certificates of all CA's within the same hierarchy as CA_u. Thus, he can eventually verify all certified public keys of other users.

Let CA_0 denote the root of the CA hierarchy. A default path from user u_1 to another user u_2 through the CA hierarchy would pass CA_{u_1} up to the root CA_0 and afterwards from CA_0 down to CA_{u_2}. The details and optimizations are given by [ISO9594-8, §7.7, §7.8].

As each public key does not necessarily have to be registered at only one KDC, neither has each item to be certified by only one CA. Again single CAs are prevented by diverse certification paths to become too powerful to be trusted by users (patients, physicians, staff etc.) any more.

8.2.3 Lifetime of certificates

The lifetime (period of validity) of certificates is subject to the security policy of MedNet. The longer certificates are valid the easier they are managed, but conversely the more chance is given to cryptanalysts to break the digital signature mechanism[34]. Hence, the longer lifetimes are selected the bigger security parameters are required which in turn leads to less effective verification (and generation) of certificates. Generation is done once per certificate, verification is done once (at most few times) per user of the certified item. Thus, a highly secure signature mechanism should be utilized in favor of long living certificates. For detailed recommendations refer to chapter 8.4.

It is proposed to update certificates by having a new certificate become valid some time before the old certificate becomes invalid. This overlapping technique [ISO9594-8, §10.2.5.1] appears to be more flexible than a strict update of certificates.

8.3 Improvements and specifications to the ISO Directory - Authentication Framework

This chapter summarizes all modifications to [ISO9594-8] recommended for MedNet. Everything not explicitly specified here is assumed to be handled as given by the ISO-standard.

---

[34] In general, a longer period of secret key usage implies more instances (representatives, couriers, machines, chip cards etc.) to be entrusted with that secret key.

1) The 1990 ISO standard uses an public-key encipherment mechanism to produce digital signatures. [ISO9594-8, §4, §8.1]. Instead, a strict distinction between confidentiality and authentication services is proposed[35]. Additionally, parametrizing both kinds of mechanisms by the same pair of secret and public key leads to dangerous interferences. One is weakening the other and vice versa.

   It is assumed that MedNet employs a public-key encipherment E and a digital signature mechanism G and that each user u holds at least one pair of keys for each of them.

   Let κ_{Eu} be the security parameter of user u for E. Let P_E, C_E be the respective sets of plaintexts and ciphertexts, and $EKey_E$, $DKey_E$ the respective sets of public encipherment and private decipherment keys of E. Finally, let

$$(ek_u, dk_u) \leftarrow keygenerate(\kappa_{Eu})$$

$$c := encrypt(ek_u, p)$$

$$p := decrypt(dk_u, c)$$

   denote key generation, encipherment, and decipherment for E.

   Analogously, let κ_G be the security parameter of user u for G. Let M_G, S_G be the respective sets of messages and signatures, and $SKey_G$, $VKey_G$ the respective sets of private signature and public verification keys of G. Finally, let

$$(sk_u, vk_u) \leftarrow keygenerate(\kappa_{Gu})$$

$$s := sign(sk_u, m)$$

$$ok := verify(vk_u, m, s)$$

   denote key generation, signing, and verifying for G.

2) The notation in [ISO9594-8, §4] has to be thoroughly modularized:

   $A\{K\} ::= K|sign(sk_A, K)$ is defined by means of a digital signature mechanism. All definitions referring to digital signatures (i.e. that of a certificate [ISO9594-8, §7.2]) should explicitly keep to the above definition instead of "simulating" a signature by means of a very specific encipherment mechanism for which hardly any other than the RSA block encipherment mechanism can be applied.

3) After a potential sender has received a public key of the intended receiver and has checked its certificate successfully, both partners should authenticate each other because two main attributes of MedNet communication are secrecy and authentication. The former requires the receiver to authenticate himself against the sender, the latter requires the sender to authenticate himself against the receiver. Thus, mutual authentication appears to be the default requirement.

   It is recommended to utilize the 2-way authentication improved according to [AnMi_90]. This protocol is recommended because three-way authentication [ISO9594-8, §9.4] has no advantage over the improved 2-way authentication.

---

35 E.g., a formal definition for digital signature systems is given by [GoMR_88]. It defines the security of a digital signature system independently from the property if a signed message is recoverable from its signature or not. To the contrary, any definition of an encipherment mechanism and thus of a public-key encipherment mechanism must define the information theoretic property that the plaintext of a corresponding ciphertext must be recoverable from that ciphertext. This settles the fundamental difference between digital signature systems and public-key encipherment mechanisms.

8.4 Recommendations

According to chapter 8.2 the digital signature mechanism utilized for producing certificates has

- to be an digital signature mechanism
- to resist attacks for the whole lifetime of a certificate and
- not necessarily to provide a very high verification speed.

According to the current cryptologic research the best choice is to utilize GMR (chapter 5.1.2). Alternatively, the RSA digital signature mechanism (chapter 5.1.3) extended by the Damgård hash mechanism can be employed.

The hash mechanism "sqr mod n" proposed by [ISO9594-8, Annex D] is not to be used because of attacks mentioned by [AnMi_90].

8.5 Establishment of symmetric keys

After u_2 has received an authentic public key of u_1 for an encipherment mechanism he could use this key to encipher messages by means of this encipherment mechanism, e.g. RSA (chapter 3.1.5). If security requirements should not demand to use RSA or performance requirements prohibited it the users u_1 and u_2 are recommended to exchange a symmetric key. This can be done in advance for a series of consecutive messages or once for each message. The decision depends upon the security policy and the categorization of the data to be transmitted.

Exchanging a symmetric key in advance should be done according to ISO9798-3 and [Fumy_90]. Otherwise, hybrid encipherment (3.1.1) enhanced by detection of modification (chapter 7) can be used.

9 SECURE HEALTH CARE DATABASES

Let a **database** consist of an Operating System (OS), a DataBase Management System (DBMS), and an Access Control Management System (ACMS). Roughly speaking, the tasks of these components are persistent storage, query evaluation and data retrieval, and enforcing some global access control strategy, respectively. Several access control strategies have been considered: discretionary, multi level security, chinese wall, personal model of data, etc. There are two stages when the ACMS plays an essential role: Whenever a user passes a query to the DBMS, the ACMS is consulted how much of the resulting answer the user is permitted to receive. Complementary, whenever a user inputs some data into the DBMS, the ACMS is responsible to prepare the input data in a way that enforces the access control strategy. The preparation might be some kind of labelling or preprocessing. Analogously, the consultation at the time of retrieval might be some kind of checking the user authorization against the data labels or it might be the inversion of the preprocessing in order to revert the data to its original form. It has long been investigated how to design and implement **secure** databases. Due to governmental initiatives [DoDS_83, DoDS_85] these investigations have focused upon the confidentiality of data rather than on detectability of modification or on authentication. A recent overview of database security is presented by [CFMS_95]. The following outlines a systematic approach to database security.

9.1 Proposed mechanisms

Specific proposals are still under consideration.

9.2 Choice of security parameters

This chapter completely depends upon the proposals of chapter 9.1.

9.3 Models of adversary control

Several possibilities of adversary control over a database have been considered in the literature. By order of increasing severity, adversaries could:

c_1 use their privileges if they are authorized users of the OS and the DBMS,

c_2 inspect the raw data which is processed or stored by the OS,[36]

c_3 inspect the raw data while it is processed by the DBMS, e.g., during evaluations of queries, by means of Trojan horses or covert channels, or

c_4 subverting the access control management system.

In the following, it is assumed that an adversary who controls a database by means of c_i also controls it by any weaker means c_j ($j < i$). Hence, only four combinations of these means are considered as interaction models. It is further assumed that there are n database available which are separately managed. Orthogonally it is distinguished if an adversary has **full control**, i.e., if he controls all instances or if he has **partial control**, i.e., if he controls at most t–1 ($t \leq n$) of the instances. In the sequel, F_i and L_i denote full and partial control at levels c_1 to c_i, respectively. For each interaction model, some counter measures are discussed. The counter measures in their turn might impose certain restrictions upon the expressive power of the query language that the DBMS can provide.

Interaction model F1

Adversaries are regarded to be authorized users of the OSs and DBMSs. I.e., users do not trust each other, but trust the databases. Active attacks are admitted since the databases are practically, and often even theoretically, unable to distinguish attacks from harmless queries.

Counter measures: Access control management and security audit at user interface level, e.g., based on user identification.

Expressive power of queries: Relational algebra.

Interaction model L2

Adversaries are regarded to be authorized users of the OSs and the DBMSs and they can inspect data stored by at most t–1 OSs.

Counter measures: DBMSs evaluate queries on plain data. ACMSs control access to plain data by enciphering it according to the access control strategy. OSs only store enciphered data.

Expressive power of queries: Relational algebra.

Interaction model F2

Adversaries are regarded to be authorized users of the OSs and the DBMSs and they can inspect data stored by all OSs.

---

[36] The inspection of data is judged to be the more severe, first, the more data an adversary can get, second, the more context information he can get for the inspected data or parts thereof.

Counter measures: DBMSs evaluate queries on plain data. ACMSs control access to plain data by enciphering it according to the access control strategy. OSs only store enciphered data [CaJü_85].

Expressive power of queries: Relational algebra without total or partial order predicates [Denn_82, ch. 3.5.2].

Interaction model L3

Adversaries are regarded to be authorized users of the OSs and the DBMSs and they can inspect data stored by at most t–1 OSs or processed by the corresponding DBMSs. For simplicity, let $f(\bullet, \bullet)$ be a function which is going to be evaluated by some DBMS d. A function $g(\bullet, \bullet)$ can be evaluated by any of the n DBMSs. How can d compute f by the help of the other DBMSs without at most any t–1 of them learning anything about the inputs nor the output of f? In general, the function f may take an arbitrary number of inputs.

Counter measures: During data input, each ACMS splits up its input data into n **shares** and passes each share to one of the n OSs. When d wants to evaluate the query f, its ACMS chooses some set of k out of the n OSs and asks them to return the result of g applied to their respective shares. The ACMS of d finally collects the shares of the result and computes the result.

In order to generate the shares of input data some **secret sharing homomorphism** φ is applied such that for each required function (operator) f of the query language with inputs a, b there is some function g such that

$$f(a,b) = \varphi^{-1}\begin{pmatrix} g\left(\varphi_{i_1}(a), \varphi_{i_1}(b)\right) \\ g\left(\varphi_{i_2}(a), \varphi_{i_2}(b)\right) \\ \vdots \\ g\left(\varphi_{i_k}(a), \varphi_{i_k}(b)\right) \end{pmatrix}$$

Each OS stores the shares of its respective DBMS. [Bena_87].

Expressive power of queries: Relational algebra.

Interaction model F3

Adversaries are regarded to be authorized users of the OSs and the DBMSs and they can inspect data stored by the OS or processed by the DBMS. For simplicity, let $f(\bullet, \bullet)$ be a function which is going to be evaluated by some DBMS d. A function $g(\bullet, \bullet)$ can be evaluated by any of the n DBMSs. How can d compute f by the help of the other DBMSs without them, even if they collude, learning anything about the inputs nor the output of f? In general, the function f may take an arbitrary number of inputs.

Counter measures: If a local DBMS wants to update a data item, its ACMS enciphers the data item and passes the result to some DBMS (e.g., some central site). Whenever some local DBMS d wants to evaluate a query f, its ACMS asks the central DBMS to return the result of g applied to the respective enciphered items. Finally, d receives the enciphered result and its ACMS can decipher it.

In order to encipher data items some uniform **privacy homomorphism** φ is applied such that for each required function (operator) f of the query language with inputs a, b there is some function g such that

$$f(a,b) = \varphi^{-1} \circ g\left(\varphi(a), \varphi(b)\right)$$

The OS only stores data enciphered by φ. [RiAD_78] proposes four privacy homomorphisms for general algebras, [DaYe_82] for algebras including the operators *join* and *projection*, [BlM1_85] for algebras including statistical operators like *average*, *standard deviation*, etc., [WaP1_86] presents a general approach. A principle problem of any *universal* privacy homomorphism φ is that all users need to share some common key(s) which enable(s) them to compute φ^{-1}, but which must be kept secret from the DBMS and the OS.

Expressive power of queries: Relational algebra without total or partial order predicates. [Denn_82, ch. 3.5.2]. According to [AhL1_87] a privacy homomorphism can easily be inverted if the operation g is chosen to be addition in ZZ, ZZ_m (for some m) , or $Z Z_2^{\mu}$ (for some μ).

Interaction model F4

Adversaries are regarded to be authorized users of the OSs and the DBMSs, they can inspect data stored by the OS or processed by the DBMS, and they can subvert any central ACMS.

Counter measures: Each user employs his individual ACMS and does not rely upon the ACMSs of the untrusted databases. Only the individual user knows from where to retrieve data that he input himself. Hence, for each individual user there is a virtual hierarchy of DBMSs. At the root his own DBMS is located whereas the nodes at lower levels are formed by the DBMSs of other users and by the shared databases. The individual ACMSs utilize the counter measures above according to the adversary model F1, F2, L3, F3 they assume for the other databases, with one essential difference. An individual ACMS no longer needs to respect the functions (or operators) provided by the individual DBMS. For example, a homomorphism property with respect to these functions is no longer necessary.

Besides the counter measures mentioned above, there are specific solutions to F4.

For multi-level security strategies, it has been proposed to encipher the data by keys from some inclusion hierarchy. I.e., the key hierarchy reflects the clearance hierarchy of the users. [DaWK_81, AkTa_83, MaAk_83, MTMA_85, Sand_88, ChTa_90].

In case of more general strategies of access control (discretionary) there are the following options: An ACMS may store the data at some trusted site (e.g., a personal device like a Smart Card). Alternatively, it may separate the input data into meaningless “atoms” and distribute them over untrusted sites [FaRa_92].

Expressive power of queries: Not quite clear. Some hints suggest that this interaction model only allows for a significantly reduced expressive power of the query language, i.e., a comfortable file system, where each user manages his own data enciphered by individual cryptographic keys.

9.4 Valuation of the results

The adversary interaction models F1, F2, L3, F3 appear to be suitable for user groups who put some minimum trust into each other and into the database. For interaction model F2 no better proposal than for F3 is known. The proposal of [CaJü_85] suffers from a significant overhead of decipherment and encipherment for every query. For interaction model L3, the secret sharing homomorphisms of [Bena_87] are promising. They allow for arbitrary

operations on data fractions and provide security against unconditionally powerful adversaries. For interaction model F3, no suitable recommendations can be made. The use of privacy homomorphisms is limited since they neither allow order predicates nor addition as operations on enciphered data. Besides, many proposed privacy homomorphisms have turned out to be insecure. E.g., [BrYa_88] and [AhL1_87] broke the proposals of [RiAD_78] and [WaP1_86], respectively.

Users who are personally liable for their actions (i.e., who cannot make some database legally liable for breaches of confidentiality) will only accept interaction model F4 (and possibly L4) as suitable.

Intuitive conclusion: "Given a security strategy, the more control adversaries have over a database, the less expressive the query language of that database can be."

9.5 Detectability of modification and non-repudiation

Naturally, the same interaction models should also be studied with respect to the other important services of confinement: detectability of modification and non-repudiation. Early papers regarded detectability as if it would simply be achieved by encipherment. They argued it would be highly unlikely that an adversary succeeded in modifying enciphered data in such a way that the modification would remain undetected after deciphering. Reliable detection, however, implies that some redundancy is inserted into the data before enciphering it. It is recommended to achieve this by operating the employed encipherment mechanism in PCBC mode (chapter 7.1). The service of non-repudiation can only be achieved by digital signatures (chapter 5). If signed results of queries are required, it follows that any input into the database must have been signed by its originator. However, it is an open research problem if and how general query evaluations can yield signed results under adversary interaction models F1, .., F3. Under interaction model F4 there are solutions, since data is only read and written, but is not subject to algebraic operations.

A DEMONSTRATION OF A SOFTWARE IMPLEMENTATION

The "prototype for encryption" subsequently referred to as **SECURE Talk** was demonstrated to the SEISMED consortium during the meeting at the University of Hildesheim on September 9, 1993. The main goal of the demonstration was to present ease of use and performance of many of the proposed cryptographic mechanisms.

The hardware configuration was formed by 4 Apple Macintoshes equipped with high resolution graphics and connected by Ethertalk. They simulated 4 medical departments: a Radiology, a Surgery, a Laboratory, and an Authentication Center. As an example i) the radiology generated an X-ray picture of 1.1 Mbyte using the graphic processor OSIRIS, ii) the radiology transmitted this picture to the surgery, iii) the surgery edited it and added some annotations and iv) finally forwarded the file to the laboratory. The transmission ii) was protected by attaching a proof of origin and enciphering the whole picture and transmission iv) was protected by only enciphering the picture. The last stage of the demonstration presented how the cryptographic keys are automatically established for such a set of workstations.

SECURE Talk provides the functionality of PGP 2.6 [Garf_95] plus automatic key management in (linked) Apple Talk™ networks. In addition, SECURE Talk provides many more cryptographic mechanisms and performs approximately twice as fast on the same Apple Macintosh platform.

It was found that the prototype is easy to use and performs sufficiently fast for many medical applications. An increase of cryptographic speed by a of factor of 3 to 4 appears to be achievable with the same cryptographic kernel.

A.1 Goal of the demonstration

SEISMED develops and implements a prototype [Bleu_94] that demonstrates secure communication utilizing only standard hardware. It shows, in particular, which mechanisms yield what performance. The most complex mechanisms are symmetric and asymmetric encipherment, message authentication codes and digital signatures.

The message of this prototype is that cryptography (and in particular asymmetric cryptography) is practical for many medical applications even if implemented in software.

The recent years showed that the computational power of hardware one can buy for one ECU increases by about 40% each year. This means that software cryptography will fulfill the requirements of more and more ambitious applications at constant cost even if algorithms are not improved. The prototype presents the operability of software implemented cryptographic mechanisms which are integrated into a simplified scenario.

This scenario is interactive, (transparent,) sequential multi-user, secure end-to-end file handling between the workstations[37] of a LAN.

It might, e.g., represent data transfer between the physicians (and/or patients) of different hospitals in possibly different regions or countries. It might, as well, represent the communication within one department of a hospital.

Interactive means that a command interpreter constantly offers a menu of communication services to the user, waits for a command, executes it and waits for the next one.

Transparent means that the **security management** is done nearly automatically. This optional feature releases the user from managing parameters and keys as much as possible. In an optimal case, one does not even notice that security services protect the communication.

Sequential multi-user means that multiple users are supported at the same workstation one after another[38]. The underlying operating system is not assumed to support a multi-user mode although this would be helpful. Users are requested to identify themselves before they gain access to the prototype. The basic version of the prototype will support user identification by passwords, future versions might replace this method by more sophisticated techniques like smart cards, biometrics etc. A dependable identification mechanism will be the key to accountability, a necessary property of a system dealing with sensitive data.

---

37 It is explicitly assumed that autonomous workstations are running in the network, not only terminals. These workstations may, but need not be equipped with hard disks, CD-drives, smart card readers, etc.

38 This is a common requirement for many workstations of a hospital information system (e.g., ward PCs).

Secure end-to-end means that the content of the communication is protected at the application layer [ISO7498-2] against unauthorized disclosure (confidentiality) and undetected, unauthorized modification (integrity). No other network layer is protected by the SEISMED prototype. This remains the task of operating system and network manager.

File handling means that SECURE Talk supports the participants of the underlying LAN in loading, storing, and securely exchanging files created by arbitrary applications. (This includes text–, code–, picture–, video–, biosignal–, soundfiles, etc.) A generic way to exchange data between applications is by file. Hence, the smallest data unit protectable by SECURE Talk is a file. Many medical applications are supposed to be supportable by this interface, i.e., their communication can be directed into files, which can be transferred by the prototype. Of course, it is not claimed that secure file handling in such a generic way covers the communication needs of *every* medical application. However, it appears unreasonable to restrict SECURE Talk to more specific data structures. For example, although SEISMED exclusively deals with data security in the health care environment, it has not identified and agreed upon specific medical data structures so far.

Beside exchanging other applications' data, SECURE Talk integrates editing and exchanging short text memos. This might be found comfortable not only by users who are already used to e-mail.

The hardware assumptions of the prototype reflect the decentralized character of a typical health care environment. It is only assumed that there is some digital, bit transparent network (e.g., Ethertalk, ISDN, etc.) connecting the workstations.

> The SEISMED prototype **SECURE Talk 1.0** is implemented for workstations that run Apple operating system 7.0 or higher and are connected by Apple Talk (Ethertalk or Locatalk). The off-line features can be demonstrated on one workstation. On-line features require a set of at least 3 workstations. Only Apple operating system 7.0 and the SECURE Talk software have to be installed on each machine. **No extra hardware, especially no crypto hardware is needed**.

Chapter 2 describes the demonstration of SECURE Talk on 9. September 1993. Chapter 3 summarizes the implementation and presents the results of the demonstration. Chapter 4 summarizes the main findings. A bibliography is appended.

A.2 Demonstration of SECURE Talk

Four Apple Macintosh workstations equipped with high resolution graphics ($1024 \times 768 \times 256$ grey scales) were connected via Ethernet (Transmission rate = 10 Mbit/s). Three of them represented a physicians workstation at some clinical departments, namely, a radiology, a surgery, and a laboratory. The fourth workstation served as an authentication center.

The demonstration proceeded in two stages, the operational stage and the initialization stage. To put emphasis on presenting usability and performance of the cryptographic mechanisms, the demonstration started with the operational stage. Finally, it was shown, how a network of Macintosh workstations had to be initialized to run SECURE Talk.

A.2.1 Operational stage

The operational stage consisted of 4 phases: At first, the radiology took an X-ray picture (Fig. A-1). This was presented by the medical image processor OSIRIS [LRGR_93]. Second, the radiologist switched to SECURE Talk, addressed the surgeon and decided to add a proof of origin to the picture as well as to encipher it. Automatically, the necessary key retrieval and cryptographic action took place and the picture file was transmitted to the surgeon. Third the surgeon got notice of an incoming file, chose to decipher it and to verify its attached proof of origin. Again, all cryptographic action was performed instantaneously and after switching to OSIRIS the surgeon had the transmitted picture in front of her. Fourth, the surgeon annotated some comments to the picture and decided to forward it to the laboratory. This time she chose to purely encipher it. Finally, the laboratory received and decipher ed the annotated picture.

Up to this point of the demonstration, all communication included reading data from the sender's fixed disk and storing it to the receiver's fixed disk. Hence, the sending time comprised read access, cryptographic operation, Ethernet transmission, and write access. The dominating link of this chain was the write access. To further investigate the performance of the cryptographic operations a RAM disk was installed and the same picture as above was enciphered and decipher ed from that RAM disk onto itself. For benchmarks see chapter 3.

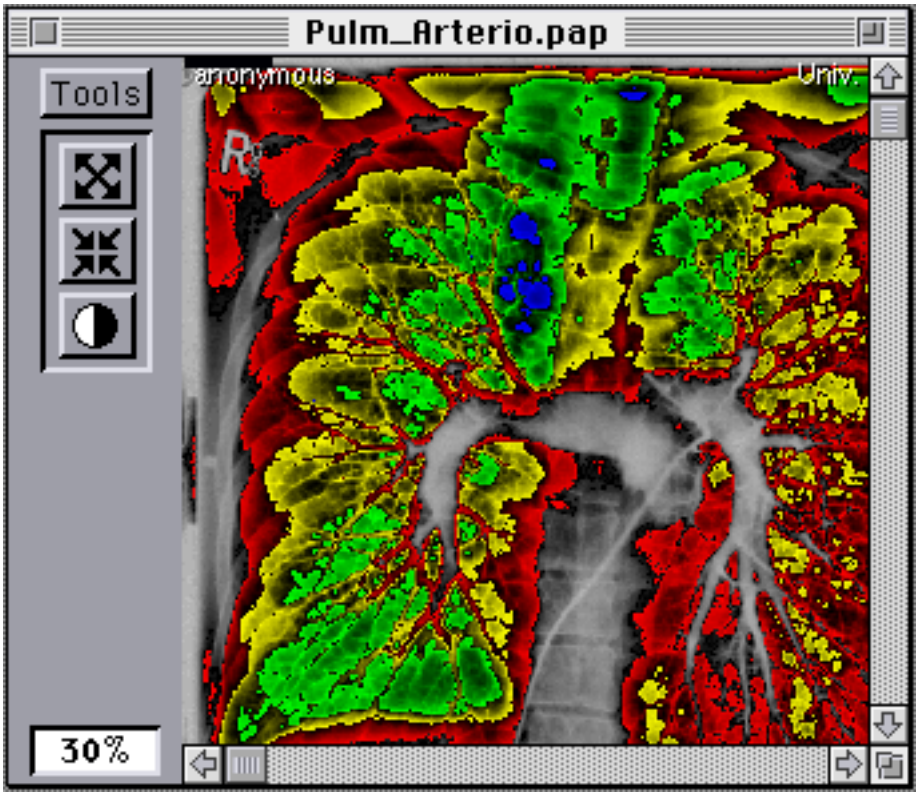

Fig. A-1 OSIRIS view of an X-ray picture

A.2.2 Initialization stage

Obviously, the operational stage relies on well generated and distributed public keys for each user participating in a SECURE Talk session. This is done right at the beginning when each user launches SECURE Talk. SECURE Talk allows every participant to generate his own secret keys[39]. Afterwards, some key distribution with simplified[40] X.509 certification is performed. The SECURE Talk Server serves as a distinguished name server, as the public key directory, and as the certification authority at the same time.

During the initialization phase each user and the SECURE Talk Server authenticate each other physically. In practice this is done by each user approaching physically the SECURE Talk Server in its role as a distinguished name server. The user is given a unique name relative to all other users registered so far. Afterwards, user and server exchange their individual verification keys by which the other party may later verify their digital signatures. This way, all subsequent communication, and in particular the key distribution, can be authenticated based on this initial (physical) authentication.

Naturally, the communication during the initial authentication has to be authenticated itself. SECURE Talk achieves this by data transfer via diskettes where the diskettes are taken physically to and from the server by the users themselves. Of course, the diskettes may be replaced by smart cards, advanced cards, etc. in a later version.

After initial authentication with each user, the operational stage begins and the SECURE Talk Server is only needed in its role as a public key directory and as a certification authority. In both roles the server runs automatically on the network. At this stage, all key management is done nearly[41] transparent to the users.

A.3 Implementation and Results

SECURE Talk 1.0 is developed under **Apple System 7.0** using an **Object oriented THINK Pascal** compiler and **MPW assembler**. The application is completely designed in a modular way. Where performance allowed it, i.e., for the graphical user interface, the advantages of objects, class hierarchies and inheritance are utilized. The cryptographic kernel of SECURE Talk is a high-speed cryptographic library **CPDB** (**Crypto Protocol Development Base**). CPDB provides DES [DES_77], G-DES [PfAß_90] and RSA [RSA_78] encipherment with several modes of operation. Combining any symmetric with any asymmetric encipherment system is supported to achieve hybrid encipherment. Several electronic signature systems like RSA and GMR [GoMR_88] are provided. The key sizes of all asymmetric systems may be defined by the user himself. The modular design of CPDB allows easy insertion of new cryptographic systems as well as improvement of existing ones. CPDB provides an operational speed of hybrid encipherment as well as of digital signatures of roughly 1,100 Kbit/s (chapter 3.5).

SECURE Talk is a standard Apple Macintosh application supporting mouse, pull down menus, dialog boxes, hot keys, etc. and thus conforms to Apple's graphical user interface conventions. The sole hardware requirement to launch SECURE Talk is a workstation

---

39 however, version 1.0 does not support to type in 512 bit integers manually. Rather it provides a pseudo random number generator for each user to do the job.

40 certification tree of fixed depth = 2

41 "Nearly" means that only some minor alerts inform the users about what is going on with the key management. But the users do not have to take explicit actions concerning key management.

running Apple OS 7.0 or higher (Performa, Macintosh, Powerbook, Centris, Quadra, etc.). The demonstration of network services, however, requires a network — at least three of the above workstations connected via Apple Talk.

A.3.1 Result of the demonstration

Of course, a demonstration presenting some transparent service aims at conflicting goals. On the one hand, it should show the whole service achieved and on the other hand it should show nothing because the service is intended to be transparent. Two aspects may show how the demonstration of SECURE Talk solved this paradox:

a) Almost every detail of using cryptographic mechanisms can be hidden from the user except the generation, storage, usage, and deletion of his secret key(s). If the user is forced to trust some special authority, subsystem, hardware, software or whatever to maintain these tasks for him he will (reasonably) not trust the whole of the integrated system.

   Hence, the golden rule: Do not try to prevent users from managing their secret keys!
   However, proper secret key management requires training.

   In contrast to this rule and completely for convenience of demonstration SECURE Talk assumes that users trust their own workstations and their application software to generate safe secret keys for them. It is, e.g. impossible for a user to type in a secret RSA key manually.[42]

b) The essential steps of the key management and progress of cryptographic mechanisms were indicated to the user by some graphical alert boxes. Of course, these visual indicators need not be integrated into applications which require complete transparency of the underlying mechanisms.

By the above means it was possible to convince the audience that cryptographic mechanisms can be sufficiently easy to use. Furthermore, the benchmarks of SECURE Talk software achieved a transmission rate with hybrid encipherment using RSA (512 bit) and DES as well as with digital signatures using RSA (512 bit) and a DES hash mechanism of 240 Kbit/s. Roughly twice the speed (447 Kbit/s) was achieved if the source and destination data were located on RAM disks which saved accessing time to fixed disks. All benchmarks were taken between an Apple Quadra 950 (MC 68040, 33 MHz) and an Apple Centris 650 (MC 68030, 33 MHz) connected via Ethertalk phase 1.

	Kbit/s	Apple OS (copying)	SECURE Talk (enciphering)
Local	disk → disk	1,921	179
	RAM → disk	2,471	215
	disk → RAM	3,052	266
	RAM → RAM	3,706	323

[42] Naturally, this might leave an audience suspecting that effectively SECURE Talk does not encipher in the strong way it is promised. However, this suspicion were not ruled out by users being able to type in their secret keys, it can only be ruled out by an evaluation of its design process, source code, compilers used, etc.

Ethertalk	disk → disk	774	240
	RAM → disk	851	263
	disk → RAM	1,266	392
	RAM → RAM	1,441	447

Fig. A-2 Performance of SECURE Talk compared to plain data transfer rates

Some closer benchmarks (Fig. A-2) were taken to investigate why SECURE Talk 1.0 proves unable to get the best out of the high cryptographic speed of the underlying cryptographic library CPDB. Up to a transmission rate of 1,100 Kbit/s the cryptographic mechanisms are not expected to be the limiting factor. But according to Fig. A-2 they are.

The following parameters were measured: Source and destination data may reside in RAM or on a fixed disk, the destination medium may be located on the same machine as the source medium or on a remote machine connected by Ethertalk. Obviously, Ethertalk slows down the data transfer rate by a factor of roughly 2.5 compared to local copying.

Column 4 presents some unexpected findings:

- Encipherment from one disk to the same disk performs some 25% slower! than encipherment over the network. Probably, this is due to some kind of mutual prevention of reading from and writing to the same fixed disk alternatingly. Hence, the benchmarks for encipherment on the local machine appear to be influenced by undetected causes.
- Encipherment over the network achieves no better transmission rate than 447 Kbit/s which is only 40% of the possible speed of 1,100 Kbit/s. Supposingly, the reason is that the actual implementation of SECURE Talk makes no use of concurrent reading and writing. Rather it performs reading, enciphering, and writing sequentially one after the other. Ideally, it were possible to put reading, encipherment, and writing into three consecutive steps of a pipeline. Let s_r, s_e, s_w denote the operational speed of the respective steps, s_{serial} be the throughput of the serial execution and $s_{pipeline}$ be the throughput of the corresponding pipeline. Then one has

$$s_{pipeline} = \min\left\{s_r, s_e, s_w\right\}$$

$$s_{serial} = \frac{1}{\frac{1}{s_r} + \frac{1}{s_e} + \frac{1}{s_w}},$$

  Consider, for example, transmission from disk to disk via Ethertalk (Fig. A-2, line 5). The Apple operating system performs reading and writing almost concurrently, hence one can assume that $s_r \approx s_w \approx 774$ Kbit/s (Fig. A-2, col. 3). With s_e = 1,100 Kbit/s one achieves s_{serial} = 286 Kbit/s. This calculation still ignores some processes that control the graphical user interface. Hence, the benchmark of 240 Kbit/s is pretty well explained.

From the above analysis one would expect an operational speed of approximately 744 Kbit/s up to 1 Mbit/s if SECURE Talk would utilized concurrent disk access optimally. It is considered that this were sufficient for many medical applications.

A.4 Summary and items of future interest

The prototype SECURE Talk basically provides the cryptographic security services of the OSI application layer at a friendly GUI. Although it were a reasonable next step, it does not attempt to integrate these services into some existing application [ECMA138].

Users of SECURE Talk can communicate confidentially and/or authentically via a local area network. They can experience more or less centralized forms of public key management and the performances of software encipherment and authentication under varying cryptographic parameters. The operational speed achieved by the current implementation is about 250..450 Kbit/s depending upon the media data is read from and written to. With some effort the current implementation could be enhanced to utilize concurrent reading from and writing to disk which would yield an operational speed of about 750..1,100 Kbit/s again depending upon the media.

Real health care information systems, of course, have to conform to a specific security policy. Thus they require additional organizational and security properties that could not be included within SECURE Talk 1.0. It remains a future task to design and elaborate them.

REFERENCES

ACGS 88 Alexi W, Chor B, Goldreich O, Schnorr C P: RSA and Rabin functions: Certain parts are as hard as the whole; SIAM J. Comput. 17/2 (1988) 194-209.

AdTa1 90 Adams C, Tavares S: The Structured Design of Cryptographically Good S-Boxes; Journal of Cryptology 3/1 (1990) 27-41.

AhL1 87 Ahituv N, Lapid Y, Neumann S: Processing Encrypted Data; Communications of the ACM 30/9 (1987) 777-780.

AkTa 83 Akl S G, Taylor P D: Cryptographic solution to a multilevel security problem; Crypto '82, Plenum Press, New York 1983, 237-249.

AMD 85 Advanced Micro Devices: Am9518/Am9568/Am28068 System Timing Controller Technical Manual; Advanced Micro Devices, Inc., 1985.

AnMi 90 I´Anson C, Mitchell C: Security Defects in CCITT Recommendation X.509 – The Directory Authentication Framework; Computer Communication Review 20/2 (1990) 30-34.

Aßma 88 Aßmann R: Effiziente Software-Implementierung von verallgemeinertem DES; Diplomarbeit am Institut für Rechnerentwurf und Fehlertoleranz der Universität Karlsruhe, Abgabe Februar 1989; ursprünglich als Studienarbeit "Effiziente MC 68000 Assembler-Implementierung von verallgemeinertem DES" in 1988 begonnen.

Aßma 89 Aßmann R: Assembler-Implementierung von modularer Langzahlarithmetik; Studienarbeit am Institut für Rechnerentwurf und Fehlertoleranz der Universität Karlsruhe 1989.

BaBK_94 Baldin T, Bleumer G, Kanne R: CryptoManager - Eine intuitive Programmierschnittstelle für kryptographische Systeme; Walter Fumy, Gisela Meister, Manfred Reitenspieß, Wolfgang Schäfer (Hrsg.) Sicherheitsschnittstellen - Konzepte, Anwendungen und Einsatzbeispiele, Proceedings des Workshops Security Application Programming Interfaces 94, Deutscher Universitäts Verlag, 17.-18. November 1994 München, 79-94.

Barr 87 Barrett P: Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a standard digital signal processor; Crypto '86, LNCS 263, Springer-Verlag, Berlin 1987, 311-323.

BeFG 89 Beller W, Frößl J, Giesler T: Spezifikation und Implementierung eines erweiterten DES-Algorithmus als VENUS-Standardzellenchip; Studienarbeit am Institut für Rechnerentwurf und Fehlertoleranz, Universität Karlsruhe (Betreuer: Oliver Haberl, Thomas Kropf), 1989.

Bena 87 Cohen Benaloh J: Secret Sharing Homomorphisms: Keeping Shares of a Secret Secret (Extended Abstract); Crypto '86, LNCS 263, Springer-Verlag, Berlin 1987, 251-260.

Berk 82 Berkovits S: Factoring via Superencryption; Cryptologia 6/3 (1982) 229-237.

Bers 93 Berson T A: Differential cryptanalysis mod 2^32 with applications to MD5; Eurocrypt '92, LNCS 658, Springer-Verlag, Berlin 1993, 71-80.

BiSh 90 Biham E, Shamir A: Differential Cryptanalysis of DES-like Cryptosystems; Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot, Israel, Technical Report CS 90-16, July 19 1990.

BiSh 91 Biham E, Shamir A: Differential Cryptanalysis of Feal and N-Hash (Extended Abstract); Eurocrypt '91, Brighton, 8-11 April 1991, Abstracts, 1-8.

BiSh3 91 Biham E, Shamir A: Differential Cryptanalysis of DES-like Cryptosystems; Journal of Cryptology 4/1 (1991) 3-72.

BiSh4 91 Biham E, Shamir A: Differential Cryptanalysis of the Full 16-round DES; Technical Report no. 708, December 1991, Computer Science Department, Technion, Haifa, Israel.

BiSh 92 Biham E, Shamir A: Differential Cryptanalysis of Snefru, Khafre, REDOC-II, LOKI and Lucifer; Crypto '91, LNCS 576, Springer Verlag, Berlin 1992, 156-171.

BKPS 93 Brown L, Kwan M, Pieprzyk J, Seberry J: Improving Resistance to Differential Cryptanalysis and the Redesign of LOKI; Asiacrypt '91, Proceedings, Fujiyoshida, November 11-14, Springer-Verlag, Berlin 1993, 36-50.

Bleu 94 Bleumer G: Security for decentralised health information systems; in B. Barber, A.R. Bakker, S. Bengtsson (ed.): Caring for Health Information: Safety, Security and Secrecy, Elsevier Science, Amsterdam 1994, 139-146.

BlGo 85 Blum ;, Goldwasser S: An Efficient Probabilistic Public-Key Encryption Scheme Which Hides All Partial Information; Crypto '84, LNCS 196, Springer-Verlag, Berlin 1985, 289-299.

BlM1 85 Blakley G R, Meadows C: A Database Encryption Scheme which Allows the Computation of Statistics Using Encrypted Data; Proceedings of the 1985 Symposium on Security and Privacy, April 22-24, 1985, Oakland, California, IEEE Computer Society, 116-122.

BoBo 92 den Boer B, Bosselaers A: An Attack on the Last Two Rounds of MD4; Crypto '91, LNCS 576, Springer Verlag, Berlin 1992, 194-203.

BoBo 94 den Boer B, Bosselaers A: Collisions for the Compression Function of MD5; Eurocrypt '93, Lofthus, Norwegen, Mai 1993, Proceedings, LNCS 765, Springer-Verlag, Berlin 1994, 293-304.

BoRu 89 Bong D, Ruland C: Optimized Software Implementations of the Modular Exponentiation on General Purpose Microprocessors; Computers & Security 8 (1989) 621-630.

BrDL 93 Brandt J, Damgård I, Landrock P: Speeding up Prime Number Generation; Asiacrypt '91, Proceedings, Fujiyoshida, November 11-14, Springer-Verlag, Berlin 1993, 440-449.

Bric 90 Brickell E F: A survey of hardware implementations of RSA; Crypto '89, LNCS 435, Springer-Verlag, Heidelberg 1990, 368-370.

BrOd 92 Brickell E F, Odlyzko A M: Cryptanalysis: A Survey of Recent Results; Gustavus J. Simmons: Contemporary Cryptology – The Science of Information Integrity; IEEE Press, Hoes Lane 1992, 501-540.

BrYa 88 Brickell E F, Yacobi Y: On Privacy Homomorphisms (extended abstract); Eurocrypt '87, LNCS 304, Springer-Verlag, Berlin 1988, 117-125.

CaJü 85 Caroll J M, Jürgensen H: Design of a Secure Relational Data Base; Computer Security: the practical issues in a troubled world, Proceedings of the Third IFIP International Conference on Computer Security, IFIP/Sec'85, Dublin, Ireland, 12-15 August, 1985, Jane B. Grimson, Hans-Jürgen Kugler (eds.), North-Holland, 1-16.

CaWe 79 Carter J L, Wegman M N: Universal Classes of Hash Functions; Journal of Computer and System Sciences 18 (1979) 143-154.

CCITT509 ISO/CCITT Directory Convergence Document: The Directory - Authentication Framework; CCITT Recommendation X.509 and ISO 9594-8, "Information Processing Systems – Open Systems Interconnection – the Directory-Authentication Framework".

ChEv 86 Chaum D, Evertse J H: Cryptanalysis of DES with a Reduced Number of Rounds; Sequences of Linear Factors in Block Ciphers; Crypto '85, LNCS 218, Springer-Verlag, Berlin 1986, 192-211.

ChRo 91 Chaum D, Roijakkers S: Unconditionally Secure Digital Signatures; Crypto '90, LNCS 537, Springer-Verlag, Berlin 1991, 206-214.

ChTa 90 Chick G C, Tavares S E: Flexible access control with master keys; Crypto '89, LNCS 435, Springer-Verlag, Heidelberg 1990, 316-322.

CFMS 95 Castano S, Fugini F G, Martella G, Samarati P: Database Security; Addison Wesley - ACM Press, 1995.

Comb 90 Comba P G: Exponentiation cryptosystems on the IBM PC; IBM Systems Journal 29/4 (1990) 526-539.

Copp_89 Coppersmith D: Analysis of ISO/CCITT Document X.509 Annex D, IBM Research Division, Yorktown Heights, June 1989

Copp 92 Coppersmith D: DES and differential cryptanalysis; appeared in an IBM internal newsgroup (CRYPTAN FORUM).

DaGV1 94 Daemen J, Govaerts R, Vandevalle J: Weak keys for IDEA; Crypto '93, LNCS 773, Springer-Verlag, Berlin 1994, 224-231.

Damg 88 Damgård I B: Collision free hash functions and public key signature schemes; Eurocrypt '87, LNCS 304, Springer-Verlag, Berlin 1988, 203-216.

DaPr 84 Davies D W, Price W L: Security for Computer Networks, An Introduction to Data Security in Teleprocessing and Electronic Funds Transfer; John Wiley & Sons, New York 1984.

DaPr 89 Davies D W, Price W I: Security for Computer Networks, An Introduction to Data Security in Teleprocessing and Electronic Funds Transfer; (2nd ed.) John Wiley & Sons, New York 1989.

DaTa 91 Dawson M H, Tavares S E: An Expanded Set of S-Box Design Criteria Based on Information Theory and its Relation to Differential-Like Attacks; Eurocrypt '91, LNCS 547, Springer-Verlag, Berlin 1991, 352-367.

DaWK 81 Davida G I, Wells D J, John B. Kam: A Database Encryption System with Subkeys; ACM Transactions on Database Systems 6/2 (1981) 312-328.

DaYe 82 Davida G I, Yeh Y: Cryptographic Relational Algebra; Proceedings of the 1982 Symposium on Security and Privacy, IEEE, 1982, Oakland, California, 111-116.

Denn 82 Denning D E: Cryptography and Data Security; Addison-Wesley Publishing Company, Reading 1982; Reprinted with corrections, January 1983.

Denn 84 Denning D E: Digital Signatures with RSA and Other Public-Key Cryptosystems; Communications of the ACM 27/4 (1984) 388-392.

DES 77 : Specification for the Data Encryption Standard; Federal Information Processing Standards Publication 46 (FIPS PUB 46), January 15, 1977.

DoDS 83 Dolev D, Dwork C, Stockmeyer L: On the Minimal Synchronism Needed for Distributed Consensus; 24th Symposium on Foundations of Computer Science (FOCS) 1983, IEEE Computer Society, 1983, 393-402.

DoDS 85 Department of Defense Standard: Department of Defense Trusted Computer System Evaluation Criteria; December 1985, DOD 5200.28-STD, Supersedes CSC-STD-001-83, dtd 15 Aug 83, Library No. S225,711.

DSS 91 : Announcing a Digital Signature Standard; Federal Information Processing Standards Publication (FIPS PUB XX), Draft, August 19, 1991.

DSS1 91 : Comments on DSS; Newgroup sci.crypt, 1991.

ECMA138 ECMA European Computer Manufacturers Association: Standard ECMA-138; Security in Open Systems – Data Elements and Service Definitions; December 1989.

ElGa 85 Taher ElGamal: A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms; IEEE Transactions on Information Theory 31/4 (1985) 469-472.

FaRa 92 Fabre J C, Randell B: An Object-Oriented View of Fragmented Data Processing for Fault and Intrusion Tolerance in Distributed Systems; ESORICS '92 (Second European Symposium on Research in Computer Security), Toulouse, LNCS 648, Springer-Verlag, Berlin 1992, 193-208.

FoPf 91 Fox D, Pfitzmann B: Effiziente Software-Implementierung des GMR-Signatursystems; Proc. Verläßliche Informationssysteme (VIS'91), März 1991, Darmstadt, Informatik-Fachberichte 271, Springer-Verlag, Heidelberg 1991, 329-345.

Forr2 90 Forré R: Methods and Instruments for Designing S-Boxes; Journal of Cryptology 2/3 (1990) 115-130.

Fox 91 Fox D: Effiziente Softwareimplementierung asymmetrischer Kryptosysteme und der zugrundeliegenden modularen Langzahlarithmetik; Diplomarbeit am Institut für Rechnerentwurf und Fehlertoleranz der Universität Karlsruhe, April 1991.

Fumy 90 Fumy W: Ein Bausteinkonzept für Schlüsselverteilmechanismen; Datenschutz und Datensicherung DuD 14/11 (1990) 573-579.

GaJo 79 Garey M R, Johnson D S: Computers and Intractability - A Guide to the Theory of NP-Completeness; W.H. Freeman and Company, New York 1979.

GaOu 91 Garon G, Outerbridge R: DES Watch: An Examination of the Sufficiency of the Data Encryption Standard for Financial Institution Information Security in the 1990's; ACM SIGSAC Review 9/4 (1991) 29-45.

Garf 95 Garfinkel S: PGP Pretty Good Privacy; O'Reilly & Associates, Sebastopol 1995

GiMS 74 Gilbert E N, Mac Williams F J, Sloane N J A: Codes which detect deception; The Bell System Technical Journal 53/3 (1974) 405-424.

GoMR 88 Goldwasser S, Micali S, Rivest R L: A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks; SIAM J. Comput. 17/2 (1988) 281-308.

Gord 85 Gordon J: Strong Primes are Easy to Find; Eurocrypt '84, LNCS 209, Springer-Verlag, Berlin 1985, 216-223.

HePe 93 van Heyst E, Pedersen T P: How to make efficient Fail-stop signatures; Eurocrypt '92, LNCS 658, Springer-Verlag, Berlin 1993, 366-377.

Herl 78 Herlestam T: Critical Remarks on some Public-Key Cryptosystems; BIT 18 (1978) 493-496.

HLMW 93 Hohl W, Lai X, Meier T, Waldvogel C: Security of Iterated Hash Functions Based on Block Ciphers; Crypto '93, Pre-proceedings, Santa Barbara, August 1993, 32.1-32.11.

HoGD 85 Hoornaert F, Goubert J, Desmedt Y: Efficient hardware implementation of the DES; Crypto '84, LNCS 196, Springer-Verlag, Berlin 1985, 147-173.

ISO7498-1 ISO: Management Framework; INTERNATIONAL STANDARD ISO IS 7498-1.

ISO7498-2 ISO: Information processing systems – Open Systems Interconnection – Basic Reference Model – Part 2: Security Architecture; INTERNATIONAL STANDARD ISO IS 7498-2; First edition 1989-02-15.

ISO8372 DIN/ISO: Informationsverarbeitung – Betriebsarten für einen 64-bit Blockschlüsselungs-algorithmus;

ISO8731 ISO 8731-1: Banking – Approved algorithms for message authentication – Part 1: DEA; ISO International Standard 8371-1; first edition 1. 6. 1987.

ISO8731 ISO 8731-2: Banking – Approved algorithms for message authentication – Part 2: Message authenticator algorithms; ISO International Standard 8731-2; first edition 15.12.1987.

ISO9594 ISO/IEC: Information Technology–Open Systems Interconnection–The directory Part 1…8; ISO/IEC International Standard 9594, 1990.

ISO9796 ISO/IEC DIS 9796: Information technology – Security techniques – Digital signature scheme giving message recovery; International Standard ISO/IEC 9796, Preliminary edition 1991-04-22.

ISO9797 ISO/IEC: Data Integrity Mechanism Using a Cryptographic Check Function Employing a Block Cipher Algorithm; ISO/IEC International Standeard 9797, 1989.

ISO9798 ISO/IEC: Information technology – Security techniques – Entity authentication mechanisms – Part 1: General model; International Standard ISO/IEC 9798-1 (1991).

ISO9979 ISO/IEC 9979-2: Data cryptographic techniques – Procedures for the registration of cryptographic algorithms; ISO/IEC Draft International Standard 9979-2 (1989).

ISO10116 ISO/IEC 10116: Information technology - Modes of operation for an n-bit block cipher algorithm; ISO International Standard, First edition 1.9.1991.

ISO10118 ISO/IEC: Hash Functions - Part 2: Hash Functions Using a Symmetric Block Cipher Algorithm; ISO/IEC Committee Draft 10118-2, 1994.

Jung 87 Jung A: Implementing the RSA Cryptosystem; Computers & Security 6/4 (1987) 342-350.

Kali3 91 Kaliski B S: An Overview of the PKCS Standards; RSA Data Security, Inc., 10 Twin Dolphin Drive, Redwood City, CA 94065, USA, June 3, 1991.

KaRS 85 Kaliski B S, Rivest R L, Sherman A T: Is the Data Encryption Standard a Group?; Preliminary Draft, April 6, 1985, paper presented at Eurocrypt '85, Linz, Austria.

KaRS 88 Kaliski B S, Rivest R L, Sherman A T: Is the Data Encryption Standard a Group? (Results of Cycling Experiments on DES); Journal of Cryptology 1/1 (1988) 3-36.

KFBG 90 Kropf T, Frößl J, Beller W, Giesler T: A Hardware Implementation of a Modified DES-Algorithm; Microprocessing and Microprogramming 30 (1990) 59-65.

LaMa 91 Lai X, Massey J J: A Proposal for a New Block Encryption Standard; Eurocrypt '90, LNCS 473, Springer-Verlag, Berlin 1991, 389-404.

LaMa2 91 Lai X, Massey J J: Markov Ciphers and Differential Cryptanalysis; Eurocrypt '91, LNCS 547, Springer-Verlag, Berlin 1991, 17-38.

LaMa 93 Lai X, Massey J J: Hash functions based on block ciphers; Eurocrypt '92, LNCS 658, Springer-Verlag, Berlin 1993, 55-70.

LeLe_90 Lenstra A K, Lenstra H W: Algorithms in Number Theory; in: Jan van Leeuwen (ed.): Handbook of Theoretical Computer Science (Vol. A: Algorithms and Complexity); Elsevier Science Publishers B.V., Amsterdam, 1990, 673-715.

LeMa1 90 Lenstra A K, Manasse M S: Factoring by electronic mail; Eurocrypt '89, LNCS 434, Springer-Verlag, Berlin 1990, 355-371.

LiPo 91 Lippitsch P, Posch R: PC-RSA, A cryptographic toolkit for MS-DOS; Proc. Verläßliche Informationssysteme (VIS'91), März 1991, Darmstadt, Informatik-Fachberichte 271, Springer-Verlag, Heidelberg 1991, 346-354.

LRGR 93 Ligier Y, Ratib O P, Girard C, Rubin P, Rejmer M: A Metropolitan Area Network for Teleradiology and Remote Expert Consultation based on ISDN; erhalten von Daniel de Roulet. Eingereicht bei CAR 93.

MaAk 83 MacKinnon S, Akl S G: New Key Generation Algorithms for Multilevel Security; Proceedings of the 1983 Symposium on Security and Privacy, IEEE, April 25 - 27 1983, Oakland, California, 72-78.

MeHe 81 Merkle R C, Hellman M E: On the Security of Multiple Encryption; Communications of the ACM 24/7 (1981) 465-467.

Meie 94 Meier W: On the Security of the IDEA Block Cipher; Eurocrypt '93, Lofthus, Norwegen, Mai 1993, Proceedings, LNCS 765, Springer-Verlag, Berlin 1994, 371-385.

MeMa 82 Meyer C H, Matyas S M: Cryptography - A New Dimension in Computer Data Security; (3rd printing) John Wiley & Sons, 1982.

Mühl 89 Mühlan M: Portierung, Optimierung und Integration eines Kryptographieprogrammes in eine Software-Umgebung; Studienarbeit am Institut für Prozeßrechentechnik und Robotik der Universität Karlsruhe, Februar 1989, Betreuer: Dipl.-Math. M. Spreng.

NaYu 90 Naor M, Yung M: Public-key Cryptosystems Provably Secure against Chosen Ciphertext Attacks; Proceedings of the 22nd Annual ACM Symposium on Theory of Computing (STOC), May 14-16, 1990, Baltimore-Maryland, ACM Press, 427-437.

OhMa 93 Ohta K, Matsui M: Differential Attack on Message Authentication Codes; Crypto '93, Pre-proceedings, Santa Barbara, August 1993, 19.1-19.11.

OrSA 91 Orup H, Svendsen E, Adreasen E: VICTOR – an efficient RSA hardware implementation; Eurocrypt '90, LNCS 473, Springer-Verlag, Berlin 1991, 245-252.

ORSP 87 Orton G A, Roy M P, Scott P A, Peppard L E, Tavares S E: VLSI implementation of public-key encryption algorithms; Crypto '86, LNCS 263, Springer-Verlag, Berlin 1987, 277-301.

PfAß 90 Pfitzmann A, Aßmann R: Efficient Software Implementations of (Generalized) DES; SECURICOM 90, 8th Worldwide Congress on Computer and Communications Security and Protection, March 13-16, 1990, Paris, 139-158.

PfAß1 90 Pfitzmann A, Aßmann R: More Efficient Software Implementations of (Generalized) DES; Interner Bericht 18/90, Fakultät für Informatik, Universität Karlsruhe 1990.

PrGV 93 Preneel B, Govaerts R, Vandewalle J: Hash functions based on block ciphers: a synthetic approach; Crypto '93, Pre-proceedings, Santa Barbara, August 1993, 31.1-31.11.

PrGV2 93 Preneel B, Govaerts R, Vandewalle J: On the power of memory in the design of collision resistant hash functions; Auscrypt '92, Gold Coast, Australia, Dezember 1992, Proceedings, Springer-Verlag, Berlin 1993, 105-121.

PrGV4 93 Preneel B, Govaerts R, Vandewalle J: Differential Cryptanalysis of Hash Functions Based on Block Ciphers; 1st ACM Conference on Computer and Communications Security, 3.-5.11.1993, Fairfax, acm press 1993, 183-188.

Pric 88 Price W L: Standards for Data Security – A Change of Direction; Crypto '87, LNCS 293, Springer-Verlag, Berlin 1988, 3-8.

Pric 90 Price W L: Progress in data security standardisation; Crypto '89, LNCS 435, Springer-Verlag, Heidelberg 1990, 620-623.

QuCo 82 Quisquater J J, Couvreur C: Fast Decipherment Algorithm for RSA Public-Key Cryptosystem; Electronics Letters 18/21 (1982) 905-907.

Resc 91 Rescrypt: Werbeblätter der ersten Kryptofirma der UDSSR; erhalten auf Crypto 91.

RiAD 78 Rivest R L, Adleman L, Dertouzos M L: On Data Banks and Privacy Homomorphisms; Foundations of Secure Computation, ed. by R.A. DeMillo, D.P. Dobkin, A.K. Jones, R.J. Lipton; Academic Press, N.Y. 1978, 169-177.

RIPE1 93 RIPE Consortium: RIPE integrity primitives Part 1: Final report of RACE 1040; Centrum voor Wiskunde en Informatica, Computer Science/Departement of Algorithmics and Architecture, Report CS-R9324, April 1993.

RIPE2 93 RIPE Consortium: RIPE integrity primitives Part 2: Final report of RACE 1040; Centrum voor Wiskunde en Informatica, Computer Science/Departement of Algorithmics and Architecture, Report CS-R9325, April 1993.

Rive 78 Rivest R L: Remarks on a Proposed Cryptanalytic Attack on the M.I.T. Public-Key Cryptosystem; Cryptologia 2/1 (1978) 62-65.

Rive 79 Rivest R L: Critical Remarks on "Critical Remarks on some Public-Key Cryptosystems" by T. Herlestam; BIT 19 (1979) 274-275.

Rive 91 Rivest R L: MD5 – New Message-Digest Algorithm (Abstract); vorgetragen auf Crypto '91, Rump Session, 13. 8. 1991.

Rive2 91 Rivest R L: The MD4 Message Digest Algorithm; Crypto '90, LNCS 537, Springer-Verlag, Berlin 1991, 303-311.

Rive4 91 Rivest R L: Letter on the Key Size of DSA; ACM SIGACT News 22/4 (1991) 43-47.

RSA 78 Rivest R L, Shamir A, Adleman L: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems; Communications of the ACM 21/2 (1978) 120-126, reprinted: 26/1 (1983) 96-99.

RSAD 91 RSA Data Security, Inc.: PKCS #1..#9: RSA Encryption Standard, Version 1.4; 10 Twin Dolphin Drive, Redwood City, CA 94065, USA, June 3, 1991.

SAFE 90 DuD Report: DES-Board für 600 Mark; Datenschutz und Datensicherung DuD 14/4 (1990) 225.

Sand 88 Sandhu R S: Cryptographic implementation of a tree hierarchy for access control; Information Processing Letters 27 (1988) 95-98.

Schn 91 Schnorr C P: Efficient Signature Generation by Smart Cards; Journal of Cryptology 4/3 (1991) 161-174.

Schn 93 Schneier B: Applied Cryptography: Protocols, Algorithms, and Source Code in C; John Wiley & Sons, New York 1994.

Sedl 88 Sedlak H: The RSA cryptography processor; Eurocrypt '87, LNCS 304, Springer-Verlag, Berlin 1988, 95-105.

Sham 79 Shamir A: How to Share a Secret; Communications of the ACM 22/11 (1979) 612-613.

Sham 85 Shamir A: Identity-Based Cryptosystems and Signature Schemes; Crypto '84, LNCS 196, Springer-Verlag, Berlin 1985, 47-53.

SHS 92 NIST: Secure Hash Standard (SHS); NIST Federal Register, 31.1.1992;

Simm1 92 Simmons G J: A Survey of Information Authentication; Gustavus J. Simmons: Contemporary Cryptology – The Science of Information Integrity; IEEE Press, Hoes Lane 1992, 379-419.

SiNo 77 Simmons G J, Norris M J: Preliminary Comments on the MIT Public-Key Cryptosystem; Cryptologia 1 (1977) 406-414.

Stel 86 Stelbrink J: Datensicherheit muß nicht teuer sein; Elektronik, Fachzeitschrift für Entwickler und industrielle Anwender 10. Januar 1986, 39-45.

VHVD 88 Verbauwhede I, Hoornaert F, Vandewalle J, De Man H: Security considerations in the design and implementation of a new DES chip; Eurocrypt '87, LNCS 304, Springer-Verlag, Berlin 1988, 287-300.

VVDJ 90 Vandemeulebroecke A, Vanzieleghem E, Denayer T, Jespers P G A: A single chip 1024 bits RSA processor; Eurocrypt '89, LNCS 434, Springer-Verlag, Berlin 1990, 219-236.

WaP1 86 Wagner N R, Putter P S, Cain M R: Encrypted Database Design: Specialized Approaches; IEEE Symposium on Security and Privacy, 1986, 148-153.

WaQu 91 de Waleffe D, Quisquater J J: Corsair: A Smart Card for Public Key Cryptosystems; Crypto '90, LNCS 537, Springer-Verlag, Berlin 1991, 502-513.

WeCa 81 Wegman M N, Carter J L: New Hash Functions and Their Use in Authentication and Set Equality; Journal of Computer and System Sciences 22 (1981) 265-279.

WeTa 86 Webster A F, Tavares S E: On the Design of S-Boxes; Crypto '85, LNCS 218, Springer-Verlag, Berlin 1986, 523-534.

Wien1 90 Wiener M J: Cryptanalysis of short RSA secret exponents; Eurocrypt '89, LNCS 434, Springer-Verlag, Berlin 1990, 372.

WiSc 79 Williams H C, Schmid B: Some remarks concerning the M.I.T. public-key cryptosystem; BIT 19 (1979) 525-538.

Zim1 86 Zimmermann P: A Proposed Standard for RSA Cryptosystems; Computer 23/9 (1986) 21-34.