

Practical cryptography - the key size problem: PGP after years

Lenka Fibíková
IEM University Essen
Essen, Germany
fibikova@exp-math.uni-essen.de

Jozef Vyskoč
VaF, s.r.o.
Bratislava, Slovakia
jozef@vaf.sk

21st December 2001

1 Introduction

The size of cryptographic keys belongs to aspects that are quite easily dealt with in theory but that have to be carefully weighted in the practical cryptography. Particularly, current as well as predicted advances in technology must be taken into account as these dictate lower bounds for the key size to ensure reasonable security even with respect to specialized machines. On the other hand too large key size, while secure in theory, brings some implementation problems especially if speed and available storage are important or limited.

The key size problem has also some relation to a purely practical problem, namely replacing a cryptosystem whose security level is no more satisfying. Particularly, if a cryptosystem still provides enough strength with respect to known analytical attacks, but advances in technology begin to make brute-force type of attacks possible, some steps need to be done to retain proper level of security. Quite obvious solution, i.e. to simply replace the old cryptosystem by another one is not always the best or even possible one as to devise a good cryptographic algorithm and especially to provide sufficient arguments on its security usually needs considerable time. But from the practical point of view even actual replacement of all installments of the old cryptosystem by new ones with minimal interruption of operations of related applications itself represents challenging problem especially in the case of cryptosystems that are of widespread use (e.g. are formal or de-facto standards). Considerably better solution seems to be the one where cryptographic algorithm used offers necessary flexibility with respect to the key size, i.e. the essential architecture of the algorithm remains the same even if the size of its key may be increased.

2 Asymmetric cryptographic algorithms

Asymmetric cryptographic algorithms provide necessary flexibility with respect to varying key sizes. On the other hand they require mathematical operations with arguments whose size directly depends on the key size, thus with the increasing key size they exhibit considerable slowdown of operations. This is especially important for applications where the speed and storage are limiting factors (e.g. smart cards).

There are three classes of asymmetric cryptographic algorithms which are nowadays commonly used or expected to be viable. Each of them is based on a specific mathematical problem, which is believed to be infeasible. These three mathematical problems are:

1. integer factorization problem — IFP (e.g. RSA)
2. discrete logarithm problem — DLP (e.g. ElGamal, DSA)
3. elliptic curve discrete logarithm problem — ECDLP (e.g. EC ElGamal, ECDSA)

Brief description of these algorithms is given in the appendix.

2.1 Security considerations

The key size problem has two aspects — theoretical and practical ones. Here we provide brief overview of known results using both theoretical and practical approaches that give us recommended lower bounds for the size of cryptographic keys. Interested reader may find more details in an excellent paper written by Lenstra and Verheul [11].

2.1.1 Theoretical Security

Strength of every cryptographic algorithm relies on the best methods that are known to solve the mathematical problem, the algorithm is based upon. For all three problems, there are special-purpose algorithms that solve the problem quickly for certain special instances. However, these cases are easy to identify, therefore it is possible to avoid them in an implementation, and we will not consider them.

Security of all the systems lies on one or more their parameters. It means that security level of the system increases by increasing the size of this parameter(s). The primary security parameter of RSA is the modulus N . The DLP has two primary security parameters — order of the underlying group and order of its subgroup — primes p and q . The primary security parameter of ECDLP is the order n of an generator of an additive group of elliptic curve points.

For the ECDLP, the best known general-purpose attack is the Pollard- ρ algorithm, whose running time is fully exponential, i.e.

$$T[n] = O(\sqrt{n}). \quad (1)$$

Thus, to solve an ECDLP instance for an elliptic curve having a $2k$ -bit parameter n takes about the same time as the exhaustive search through all k -bit keys of a symmetric cryptosystem (assuming that there is no better attack on the symmetric cryptosystem than the exhaustive search).

Pollard- ρ algorithm is also the basis for the best known general-purpose attack on the subgroup DLP. Therefore, the size of q in the DLP systems should have about the same size as the order n in ECDLP to get the same level of security. [10]

The best known general-purpose attack on the IFP (i.e. on N) as well as on the discrete logarithm problem (i.e. on p) are based on the General Number Field Sieve Method which runs in sub-exponential time, i.e.

$$T[x] = O(\exp((c + o(1)) \cdot \ln^{\frac{1}{3}} x \cdot \ln^{\frac{2}{3}} \ln x)), \quad (2)$$

where $c = (\frac{64}{9})^{\frac{1}{3}} \approx 1.923$, x equals to N for the IFP and p for the DLP, and the $o(1)$ term goes to zero as x goes to infinity [13]. This means that, to obtain the same level of security, it has to hold

$$|p| = |N|, \quad |q| = |n|, \quad |n| = 2k, \quad (3)$$

and

$$|n| = 2(c + o(1)) \cdot \ln^{-\frac{2}{3}} 2 \cdot |N|^{\frac{1}{3}} \cdot \ln^{\frac{2}{3}}(|N| \ln 2) \quad (4)$$

where $|\cdot|$ is the length of a parameter in bits, and k is key size of a symmetric cryptosystem.¹

The following table compares the equivalent security level for some commonly considered key sizes [10]:

symmetric schemes (key size in bits)	RSA (n in bits)	DLP		ECC (n in bits)
		(p in bits)	(q in bits)	
56	512	512	112	112
80	1024	1024	160	160
112	2048	2048	224	224
128	3072	3072	256	256
192	7680	7680	384	384
256	15360	15360	512	512

2.1.2 Practical Security

To discuss practical security of cryptographic algorithms one has to be very careful as there are too many fuzzy factors (e.g. opponent's budget, time, workforce, skills, etc.); moreover technological advances make it a rather moving target. We favor an approach by Lenstra and Verheul [11] which provides the following estimates.

In 1977 the Data Encryption Standard (DES) was introduced as standard for protection of sensitive unclassified data. It was stipulated to be reviewed every five years, thus we can assume that it provided adequate security for commercial use in 1982.

Hypothesis 1 [11] *56 bit keys were believed to provide adequate security in year 1982.*

A rough estimate of increasing computing power over time can be obtained by applying an empirical rule, called Moore's Law, which states that the computing power available for a given cost doubles every 18 months.

Hypothesis 2 [11] *Every 18 months the amount of computing power and random access memory one gets for a given cost doubles.*

This means that to obtain the same level of security over time, symmetric keys has to be about 7 bits longer every ten years. More precisely, taking into account the Hypothesis 1, in year y symmetric keys should be $2(y - 1982)/3$ bits longer than in 1982.

Besides the increasing computing power we have to consider also how budgets may change over time. The US Gross National Product shows a trend of doubling every ten years [11], thus we can assume that also budgets of attackers have similar progress.

Hypothesis 3 [11] *The budgets for breaking cryptographic keys doubles every 10 years.*

This means that we need to add one bit to the key size every 10 years. More precisely, taking into account the Hypothesis 1 in year y , the key size should increase $(y - 1982)/10$ bits concerning the increasing budgets as compared to the year 1982.

It is not possible to estimate what cryptanalytic development will take place in following years. However, regarding trends in cryptanalysis from year 1970 it is reasonable to assume that cryptographic findings will not have dramatic impact on the current state of cryptography.

¹Note that parameter p in 3 is the primary security parameter of the DLP, and differs from parameters p mentioned in appendix for the other two classes.

Hypothesis 4 [11] *For all systems we assume that no substantial cryptanalytic development will take place.*

Note that [11] introduces more sophisticated method to get the adequate security level over time, however, their results show that the key size we present here may be decreased and the difference in key size between ECC and the other two classes of cryptosystems is even bigger (favoring ECC).

Summarizing the key size increase according to the Hypothesis 1–4, the key size providing adequate security in year y is

$$56 + \frac{2(y - 1982)}{3} + \frac{y - 1982}{10} = \frac{23}{30}y - 1463533$$

and consequently the key sizes presented in the previous subsection may be considered as secure until the following years:

symmetric key size	56	80	112	128	192	256
year	1982	2013	2055	2075	2159	2242

Note that by combining these estimates with the table at the end of subsection 2.1.1, one can derive respective estimates for asymmetric algorithms.

In order to avoid an attack, an important question is how difficult it is to derive the result after running the attack only partially. Consider a partial attack which executes only an $\frac{1}{x}$ fraction of the full attack on a system. In case of the exhaustive search, the probability that an attacker finds the key after running only $\frac{1}{x}$ of the full attack is $\frac{1}{x}$. The General Number Field Sieve Method for both the IFP and DLP is effective only if it runs to completion; there is no chance to obtain the result sooner. Thus the probability to get the result after running only $\frac{1}{x}$ of the full attack is zero. There is the Elliptic Curve Method which is asymptotically worse than General Number Field Sieve Method method, but it produces a nonzero probability in an partial attack. However, the probability of getting the result by this attack may be considered to be negligible. The success probability of Pollard- ρ method (for ECDLP and SDLP) is proportional to the square of the fraction of the work performed, i.e. the probability to obtain the result after running only $\frac{1}{x}$ of the full attack is $\frac{1}{x^2}$. This implies that on average incomplete attacks cannot be expected to pay off. [11]

2.1.3 Forward Secrecy

If an adversary reveals the current encryption key, he can read all messages encrypted by this key. This problem is usually solved by using ephemeral keys, which are generated anew for each session and destroyed when the session is ended. If an adversary obtains one session key, he is not able to decrypt messages from any other session. This security attribute is called forward secrecy [10]. Since the time consuming process of generation of elliptic curves in ECC may be run in advance and then the elliptic curve may be used for generation of many private/public key pairs, the generation of ephemeral keys is no problem in ECC. The same holds also for DLP systems. The key generation process in IFP is much more complex which limits the possibility of changing keys in IFP systems more often.

2.2 Efficiency

Efficiency of asymmetric cryptosystems is based on three factors [6]:

1. computational overheads, i.e. how much computation is required to perform cryptographic transformations;
2. size of key parameters, i.e. how many bits are required to store key pairs and their domain parameters;
3. bandwidth, i.e. how many bits must be communicated to transfer an encrypted message or a signature.

In this section we compare implementation efficiency of the above mentioned classes of cryptosystems on the currently accepted level of security — on the level of 80-bit symmetric keys, i.e. 1024-bit IFP and DLP systems and 160-bit ECC.

2.2.1 Computational overheads

In every system, some computational savings can be made. In RSA, a short public exponent can be employed (although this has security risks) to speed up encryption and signature verification. In both DLP systems and ECC, a large part of encryption and signature generation can be precomputed. This means that the comparison of the systems strongly depends on their implementation, and consequently there are different results.

Robshaw and Yin present in [14] the following table:

	RSA with 1024-bit N , $e=216+1$, and CRT	DLP systems with 1024-bit prime	ECDSA or ECES ^a over F_p with 160-bit p
encryption	17	480	120
decryption	384	240	60
signing	384	240	60
verification	17	480	120

^aECES = Elliptic Curve Encryption Scheme defined in IEEE P1363 Standard for Public-Key Cryptography.

Figures in the table are the number of time units required to complete the given operation under the assumption that one 1024-bit modular multiplication requires one unit of time.

On the other hand Certicom claims in [6] that the most efficient implementations of ECC are an order of magnitude (roughly 10 times) faster than either RSA or DSA. They admit making RSA comparable with ECC in encryption and signature verification (but not in decryption and signature generation) by using short public exponent.

2.2.2 Size of key parameters

The following table compares sizes of key parameters for the mentioned systems (RSA, ElGamal/DSA, EC ElGamal/ECDSA).

	Domain Parameters (in bits)		Public key (in bits)		Private key (in bits)	
RSA	—		n (1024) e (64)	1088	p (512) q (512) d (1024)	2048/ 1024^a
DLP	p (1024) q (160) d (1024)	2208	y (1024)	1024	d (160)	160
ECC	E.a (160) E.b (160) p (160) P (161) n (160)	801	Q (161)	161	d (160)	160

^aIt is not necessary to store parameters p and q.

From the table, it is easy to see that key parameters of ECC are much shorter than in the other two classes of cryptosystems.

2.2.3 Bandwidth

Public key cryptosystems are commonly used to transport session keys for symmetric key cryptosystems and to generate signatures. We will consider encrypting symmetric keys on the same level of security, i.e. 80-bit symmetric keys. The following table compares size of ciphertexts and signature blobs for the mentioned systems:

	RSA	DLP	ECC
ciphertext	1024	2048	481/321
signature	1024	320	320

We present two sizes of ciphertext in ECC. The first one is the size of full encryption as described in appendix A.3. This requires input of 2 times 160 bits. In case of encryption of an 80-bit key, the second part of the input does not contain data, thus we can skip it and spare 160 bits.

In summary, the ECC is more efficient than other two classes of public-key cryptosystems in terms of size of key parameters and bandwidth. Furthermore, as follows from the previous section the difference in these parameters (and presumably also in computational efficiency) between ECC and the other classes quickly grows with the increasing demand on security.

3 Symmetric cryptographic algorithms

For many symmetric cryptographic algorithms, especially for those for block ciphers, the key size is fixed by design. Consequently they do not provide such key size flexibility as asymmetric algorithms. Changing view towards this property has been recently illustrated by basic requirements for DES replacement, where AES candidates were required to provide limited scalability, i.e. to support three different key lengths. Other approaches were studied as well and some scalable block ciphers offering even more flexibility were proposed in [5]. We will not go into details here, but it is sufficient to say that currently

there are symmetric algorithms that allows one, if necessary, to increase key size without radical changes in the architecture of the algorithm. Concerning their security, at least in the case of AES (or in fact all 5 final candidates from which AES was chosen) there are probably no doubts that it offers enough security for a couple of years while key size remains within reasonable limits.

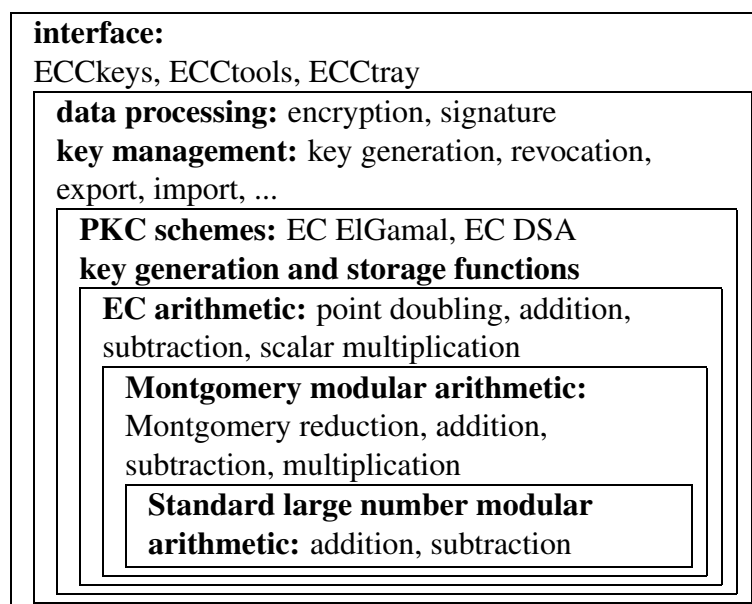
4 PGP

PGP is perhaps the most popular cryptographic package that undoubtedly made protection provided by cryptography available to masses. As time passes, however, the cryptographic algorithms used as its basic building blocks need to be reevaluated in the light of cryptographic advances made from the time of PGP creation. As shown in sections 2.1.1 and 2.1.2, using PGP in its original form, i.e. with RSA, means that to provide acceptable level of security one has to use quite large keys today and considerably larger in the future. As mentioned above, such key sizes might be a problem if speed and available storage are important and limited.

In our project we have implemented system similar to PGP but taking into account the current state of cryptography, i.e. new algorithms as well as certain emphasis on the scalability of algorithms with respect to the key size. Particularly, as a possible improvement of PGP we have implemented its EC version using the Menezes-Vanstone variant of the EC ElGamal algorithm for public key encryption, EC DSA for signing, and two newer symmetric cryptographic algorithms — the new encryption standard AES and a new fully scalable cryptographic algorithm called iterative TST (see [5]).

5 Our project

While we have tried to keep user interface similar to that provided by PGP, "innards" of our system are of course quite different. The overall description of the structure of our system is on the following picture:



In what follows we briefly describe individual modules of our system.

5.1 Large number modular arithmetic

In order to be able to avoid brute force types of attacks, elliptic curves used in cryptography must lie on large finite fields. It means that we need an implementation of large numbers whose size is much larger than the basic computer word, as well as basic operations on them (addition, multiplication, etc.) in modular arithmetic. The least efficient operation in the standard implementation is multiplication: By multiplying two numbers, size of the result doubles and then division operation has to be applied in order to get an element of the underlying finite field. Much efficient multiplication is offered by Montgomery representation of field elements. Using this representation, no division operation is needed and the multiplication and reduction steps can be interleaved, thus the space is saved too. The addition and subtraction operations work in the standard way. In our implementation we used BIGNUM library of OpenSSL for the standard implementation. We have implemented the Montgomery arithmetic as presented in [2], and formally proved correctness of our implementation.

5.2 Elliptic curve arithmetic

An elliptic curve cryptosystem, of course, fundamentally depends on a proper elliptic curve. Consequently, every implementation of a cryptosystem based on elliptic curves must cope with the problem of selecting/generating of good elliptic curves.

Having a good elliptic curve, the private key generation process reduces to the generation of a large random number with no further conditions. Moreover, the elliptic curve is a part of the domain parameters, i.e. it is publicly known, and may be reused by more users. It means that the process of generation may be run once and then many private/public key pairs may be generated.

There are some possibilities, how to get a good elliptic curve:

1. Generate an EC at random. ANSI X9.62 standard [1] introduced an algorithm for generation of elliptic curves verifiably at random.
2. Let an authority generate an EC. If the authority generates it following the above mentioned algorithm, everyone can later verify that it is a good, random curve. Such an authority may be a specialized department in an organization, a certification authority, or an independent provider like Kurvenfabrik (see below).
3. Use an EC from a set provided by some standards. NIST FIPS 186-2 [7] and SEC2 [15] standards offer precomputed elliptic curves of different sizes generated verifiably at random.

Our system provides three hard-coded elliptic curves — a 160 bit curve from [2]² and the 192 and 256 bit curves from [7]. Furthermore, we enable users to choose their own elliptic curves and load their domain parameters during the key pair generation.

In our project we decided not to implement an EC generation algorithm; we recommend to use the Kurvenfabrik (www.kurvenfabrik.de) to everyone who would like to have their own strong elliptic curve. The Kurvenfabrik project is maintained by the group of professor Frey, who is intensively working in the elliptic curves area and is known by his theoretical results. Thus, one may assume that also in future new results in ECC will be considered in the generation of elliptic curves by this program.

²FIPS 186-2 does not provide any elliptic curve of 160 bits and in time of implementation SEC2 was not yet available.

Of course, relying on someone else to provide a good elliptic curve might not be satisfying solution for somebody. For truly paranoid wishing to generate their own curve we refer to [1] and [2] for how-to ideas and proofs.

An elliptic curve over a prime finite field F_p is given by an equation:

$$E : Y^2 = X^3 + aX + b,$$

where both a and b are from the underlying field. Elliptic curve points are solutions (x, y) in $F_p \times F_p$ to the equation E along with a special (zero) point — called point at infinity, and denoted by \mathcal{O} . Points on an elliptic curve create an additive group with addition operation defined as follows:

Let P and Q be two distinct points on the elliptic curve E . The straight line joining these points intersects the curve at one other point, say R . The reflection of R in the x-axis is called $P + Q$ (i.e. $R = -(P + Q)$).

This definition does not enable addition of a point to itself, thus a special operation of doubling of a point has to be defined. Calculation of $P + P$ works similarly to the addition of two different points with the only difference that the tangent to the curve at P is constructed. If the tangent is vertical, it intersects the curve at the point at infinity, i.e. $P + P = \mathcal{O}$.

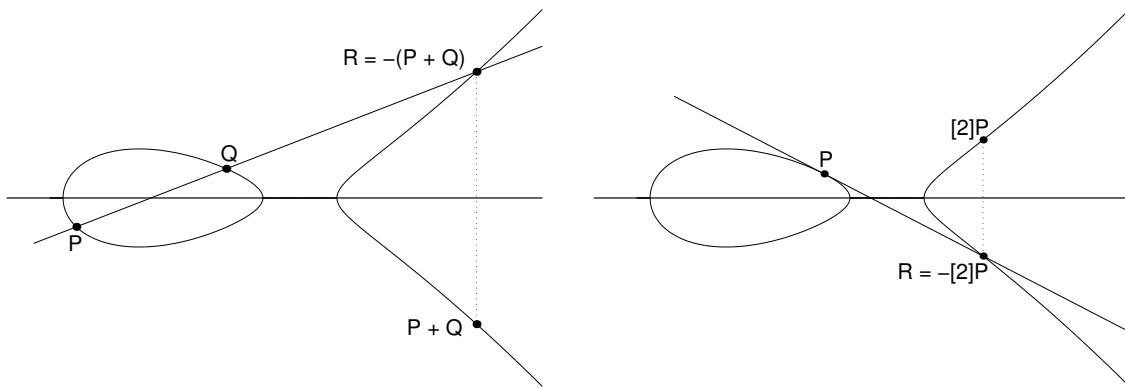


Figure 1: Point addition and doubling

Standard DLP cryptosystems are based on multiplicative groups with the main operation of exponentiation. In ECC, the multiplicative group is replaced by the additive group of elliptic curve points and exponentiation operation by scalar multiplication of a point (i.e. calculation of $g^k = g \cdot g \cdot \dots \cdot g$ for a generator g of a multiplicative group is replaced by calculation of $[k]G = G + G + \dots + G$ for a generator point G of an additive group of elliptic curve points). Thus, the computational performance of cryptographic protocols based on elliptic curves strongly depends on efficiency of the scalar multiplication. We have implemented efficient algorithms for arithmetic on elliptic curve points ([2], [13]), and also formally proved their correctness.

5.3 PKC and conventional schemes, key generation

Having implemented basic EC arithmetic we are now prepared to implement a public key cryptosystem on it. Most of known EC-based public key cryptosystems are based on EC variant of ElGamal PKC; here we made use of Menezes and Vanstone approach described in [16] for encryption and ECDSA [7] for signature generation (for details see the appendix A.3).

As we mentioned in previous subsection, generation of raw key pairs for ECC is a very simple process: To generate a private key we need to generate a long random number of the same size as the order of the group generator G (see appendix A.3 and 2.2). The corresponding public key is then calculated as scalar multiplication of the group generator by the private key.

For symmetric encryption we used implementation of the AES available at [8] and implementation of the iterative TST available at [4].

5.4 The cryptlib library

The large number modular arithmetic, elliptic curve arithmetic and cryptographic protocols form the low level of our system. We have combined their implementation with the cryptlib library [9] which provides all other functionality necessary for building a cryptographic system, like implementation of hash functions (for signature generation), and a random data management system (which provides the cryptographically strong random data used to generate session keys and public/private key pairs — the random data pool is updated with unpredictable process-specific information as well as system-wide data).

The library is built around a security kernel. This kernel provides the interface between the outside world and the architecture's objects (intra-object security) and between the objects themselves (inter-object security). Each object is contained entirely within the security perimeter, so that data and control information can only flow in and out in a controlled manner, and objects are isolated from each other within the perimeter by the security kernel. [9]

Furthermore, the cryptlib library provides an interface to the encryption and authentication functions which allows an easy implementation of higher levels of the system, and storage functions for private keys and X.509 certificates.

5.5 Data processing and key management

Data processing and key management functions are built on the top of the enhanced cryptlib library and together form the next level of the system.

Key management functions enables one to generate public/private key pairs using one of the hard-coded elliptic curves or an elliptic curve defined by domain parameters (see subsection 5.2), revocation of private keys, import and export of public keys. Since unexperienced users tend to export and send their private key (except of the public key) to their communication partners [17], we do not allow to export private keys.

There are three basic data processing functions:

1. conventional encryption: data are compressed and encrypted using CBC mode of either AES or iterative TST with a key generated from a passphrase provided by a user;
2. PKC encryption: data are compressed and encrypted using CBC mode of either AES or iterative TST with a random session key, the session key is encrypted using EC ElGamal with a recipient's public key, and appended to the encrypted message (in both cases compression may be omitted, but it is recommended and default);
3. data signature: a signature blob is created from data using EC DSA with RIPEMD-160 hash function, and appended to the original data.

The system allows also combination of encryption and signing, as well as generation of detached signatures.

5.6 Interface

On the top of the system is the user interface. It consists of three Windows programs — ECCkeys, ECCtools, ECCtray — whose work and look mimics standard PGP programs. ECCkeys provides interface to the key management functions. ECCtools allows to encrypt/decrypt and sign/verify data stored in files (all types of data files are allowed). ECCtray works like ECCtools but only with text messages stored in the Clipboard or in a simple window.

5.7 Certification Authority vs. Web of Trust

One of the key problems in practical use of public key cryptography is the problem of how to ensure/verify authenticity of public keys. Nowadays there are two, principally different, approaches to cope with this problem, namely Certificate authorities and the Web of Trust (popularized especially by PGP). Both approaches have their pros and cons but neither provides completely satisfying solution. The worst problem in the both systems is the key revocation, but this is the well known Achilles' Heel of the public key cryptography in general, and it seems that this problem will not be properly solved in the near future.

For these reasons we have decided not to bind ourselves to any of these two key management systems. In fact, our system is flexible enough to implement any of them. Public keys are stored as X.509 certificates with local storage of revoked keys. Thus, for the full CA key management one has yet to implement necessary services (namely Certification Service, Directory Service, and Certificate Revocation List). Similarly, for the full Web of Trust, meta-introducing and partial trust have to be implemented.

6 Conclusions

Elliptic curve cryptography, while known in theory for a couple of years, still suffer from the lack of practical experiences. It is our belief that implementing PGP-like system based on elliptic curves cryptography and making it available for others could help to overcome that shortage. In the meantime our implementation of elliptic curve arithmetic turned out to be appropriate for, and was included into OpenSSL project (see www.openssl.org)

References

- [1] ANSI X9.62, Public key cryptography for the financial services industry: The Elliptic Curve Digital Signature Algorithm (ECDSA). <http://www.x9.org/>, 1998.
- [2] I. F. Blake, G. Seroussi, and N. P. Smart. *Elliptic Curves in Cryptography*. Cambridge University Press, July 1999.
- [3] Dan Boneh and Ramarathnam Venkatesan. Breaking RSA may not be equivalent to factoring. In *Advances in Cryptology - Crypto '98*, pages 59–71, Berlin, 1998. Springer-Verlag. Lecture Notes in Computer Science Volume 963.
- [4] Valer Canda. TST implementation. <http://www.exp-math.uni-essen.de/~valer/work.htm>.
- [5] Valer Canda and Trung van Tran. Scalable block ciphers based on Feistel-like structure. Technical report, Institute of Experimental Mathematics, <http://www.exp-math.uni-essen.de/preprints/2001/Iterative.ps>, 2000.

- [6] Certicom Corp. Current public-key cryptographic systems. http://www.certicom.com/resources/w_papers/w_papers.html, July 2000.
- [7] FIPS 186–2, Digital Signature Standard (DSS). <http://csrc.nist.gov/publications/fips/>, 2000.
- [8] Brian Gladman. AES implementation. http://fp.gladman.plus.com/cryptography_technology/rijndael/.
- [9] Peter Gutmann. *cryptlib*. <http://www.cs.auckland.ac.nz/~pgut001/cryptlib/index.html>, September 2001.
- [10] Don B. Johnson. ECC, future resiliency and high security systems. http://www.certicom.com/resources/w_papers/w_papers.html, March 1999.
- [11] Arjen K. Lenstra and Eric R. Verheul. Selecting cryptographic key sizes. In *Public Key Cryptography*, pages 446–465, 2000.
- [12] Ueli M. Maurer. Towards proving the equivalence of breaking the Diffie-Hellman protocol and computing discrete logarithms. In *Advances in Cryptology - Crypto '94*, pages 271–281, Berlin, 1998. Springer-Verlag. Lecture Notes in Computer Science Volume 839.
- [13] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. The CRC Press series on discrete mathematics and its applications. CRC Press, 2000 N.W. Corporate Blvd., Boca Raton, FL 33431-9868, USA, 1997.
- [14] M. J. B. Robshaw and Y. L. Yin. Elliptic curve cryptosystems. http://www.rsa.com/rsalabs/ecc/elliptic_curve.html, June 1997.
- [15] SEC 2: Recommended elliptic curve domain parameters. http://www.secg.org/secg_docs.htm, 2000.
- [16] D. R. Stinson. *Cryptography Theory and Practice*. CRC Press, Boca Raton, 1995.
- [17] Alma Whitten and J. D. Tygar. Why Johnny can't encrypt: A usability evaluation of PGP 5.0. In *Proceedings of the 8th USENIX Security Symposium*, August 1999.

A Appendix

A.1 RSA

RSA is one of the first public key cryptosystem, proposed in 1977 and named after its inventors R. Rivest, A. Shamir, and L. Adleman. The security of RSA is based on the integer factorization problem (i.e. on the mathematical difficulty of calculating two large primes from their composition). However, it has not been proved that breaking of RSA is equivalent to this problem, and there are some indicia that it is not [3]. It means that RSA might be in future broken in other way than by factoring the modulus, but it does not mean that breaking RSA is not intractable.

Name:	RSA
Underlying problem:	factoring of large integers
Key generation:	select two large primes p and q ; set the modulus to $N = pq$; select integers e and d such that $ed \equiv 1 \pmod{(p-1)(q-1)}$

Public key:	modulus N and encryption key e
Private key:	decryption key d
Prim. sec. parameter:	p (min. 1024 bits)
Message:	$m \in F_N^*$
Confidentiality:	
Encryption:	$c = m^e \bmod N$
Ciphertext:	$c \in F_N^*$
Decryption:	$m = c^d \bmod N$
Authenticity:	
Signing:	$s = m^d \bmod N$
Signature:	$s \in F_N^*$
Verification:	$m \stackrel{?}{=} s^e \bmod N$

A.2 ElGamal/DSA

The discrete logarithm problem is defined as follows: Given x and y from a multiplicative group \mathbb{Z}_p^* to find such a number a that $x^a = y \pmod{p}$. The first cryptosystem of this class was ElGamal's encryption and signature scheme invented in 1984. The signature scheme was later improved into Data Signature Algorithm.

The subgroup discrete logarithm problem (SDLP) is like the traditional DLP, except that the group generator g generates only a relatively small, but sufficiently large, subgroup of the multiplicative group \mathbb{Z}_p^* . Clearly, if we know how to calculate the DLP then also the SDLP can be solved.

The ElGamal encryption scheme, DSA, as well as most other cryptographic schemes based on DLP (ECDLP — see A.3 — as well) are based on the Diffie-Hellman assumption which states that given g^u and g^v , where u, v are drawn randomly from $\{1, \dots, p(q)\}$, it is hard to compute g^{uv} . Unlike the RSA, there are some indicia, that the Diffie-Hellman assumption is equivalent to the computation of discrete logarithm in the underlying multiplicative group [12].

Name:	ElGamal/DSA
Underlying problem:	discrete logarithm problem in a multiplicative group \mathbb{Z}_p^* subgroup discrete logarithm problem in a subgroup \mathbb{Z}_q^* ($q < p$)
Key generation:	select a large prime p ; select a generator g of \mathbb{Z}_p^* ; choose a number d and calculate $y = g^d \bmod p$
Domain parameters:	modulus p and group generator g
Public key:	group element y
Private key:	integer d
Prim. sec. parameters:	p (min. 1024 bits) q (min. 160 bits)
Message:	$m \in \mathbb{Z}_p^*$
Confidentiality (ElGamal):	
Encryption:	$c_1 = g^k \bmod p$ (for a random integer k) $c_2 = m * y^k \bmod p$ $c = (c_1, c_2)$
Ciphertext:	$c \in \mathbb{Z}_p^* \times \mathbb{Z}_p^*$
Decryption:	$m = (c_1^d)^{-1} * c_2 \bmod p$
Authenticity (DSA):	
Signing:	$s_1 = g^k \bmod p \bmod q$ (for a random integer k)

$$\begin{aligned}
& s_2 = (d * s_1 + m) * k^{-1} \bmod q \\
& s = (s_1, s_2) \\
\text{Signature:} & s \in F_q \times F_q \\
\text{Verification:} & e_1 = m * s_2^{-1} \bmod q \\
& e_2 = s_1 * s_2^{-1} \bmod q \\
& g^{e_1} * y^{e_2} \bmod p \stackrel{?}{=} s_1
\end{aligned}$$

A.3 EC ElGamal/EC DSA

Elliptic curve cryptography (ECC) was first proposed in 1985 independently by Neal Koblitz and Victor Miller, and is based on the elliptic curve discrete logarithm problem (ECDLP). The ECDLP is an alternative of the standard discrete logarithm problem (DLP) applied on a group of elliptic curve points. It says: Given two points P and Q on an elliptic curve E , determine the integer k ($0 \leq k < n$), such that $Q = [k]P$, provided that such an integer exists.

Name:	EC ElGamal/ECDSA
Underlying problem:	elliptic curve discrete logarithm problem
Key generation:	select an elliptic curve E over a finite field \mathbb{Z}_p ; select a point P with a sufficiently large order n ; choose a random integer d and calculate $Q = [d]P$
Domain parameters:	elliptic curve E , modulus p , point P , order n
Public key:	point Q
Private key:	integer d
Prim. sec. parameter:	n (min. 160 bits)
Message:	$m \in \mathbb{Z}_p^* \times \mathbb{Z}_p^*$
Confidentiality (EC ElGamal):	
Encryption:	choose a random integer k from \mathbb{Z}_n^* $kG = [k]G$ $(q_1, q_2) = [k]Q$ $c_1 = q_1 m_1 \bmod p$ $c_2 = q_2 m_2 \bmod p$ $c = (kG, c_1, c_2)$
Ciphertext:	$c \in E \times \mathbb{Z}_p^* \times \mathbb{Z}_p^*$
Decryption:	$(q_1, q_2) = [d]kG$ $m_1 = c_1 q_1^{-1} \bmod p$ $m_2 = c_2 q_2^{-1} \bmod p$ $m = (m_1, m_2)$
Authenticity (ECDSA):	
Signing:	choose a random integer k from \mathbb{Z}_n^* $(x, y) = [k]G$ $s_1 = x \bmod p$ $s_2 = (d * s_1 + m) * k^{-1} \bmod n$ $s = (s_1, s_2)$
Signature:	$s \in \mathbb{Z}_n^* \times \mathbb{Z}_n$
Verification:	$e_1 = m * s_2^{-1} \bmod n$ $e_2 = s_1 * s_2^{-1} \bmod n$ $(x, y) = e_1 G + e_2 Q$ $x \bmod n \stackrel{?}{=} s_1$