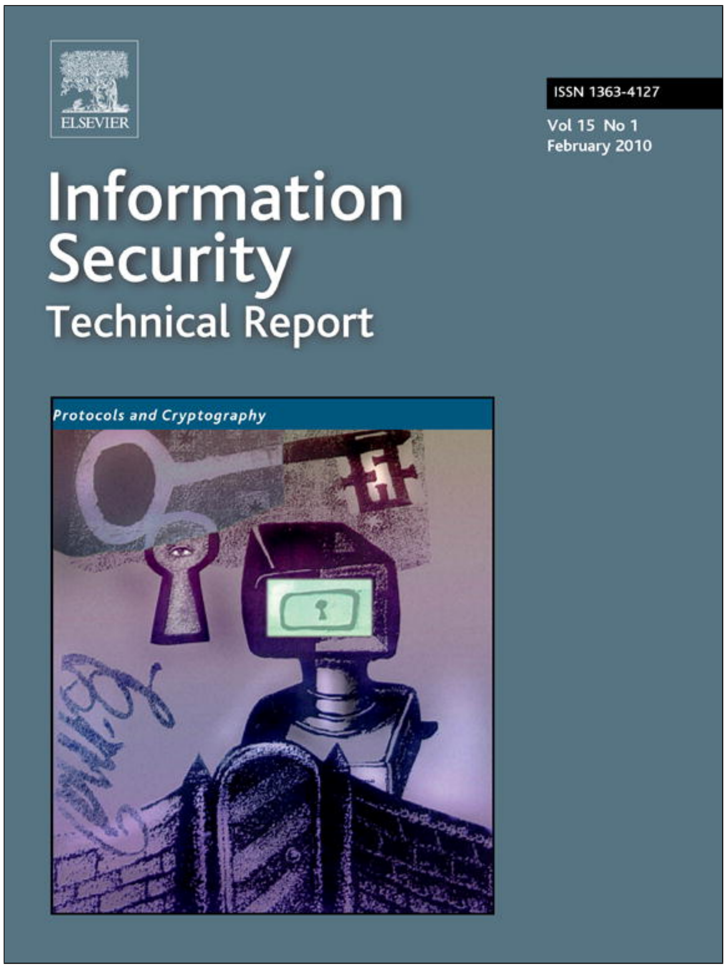
ISSN 1363-4127
Vol 15 No 1
February 2010
Information Security Technical Report
Protocols and Cryptography
ELSEVIER

INFORMATION SECURITY TECHNICAL REPORT 15 (2010) 21–27

available at www.sciencedirect.com

ScienceDirect

www.compseconline.com/publications/prodinf.htm

Information Security Technical Report

Choosing key sizes for cryptography

Alexander W. Dent

Information Security Group, University Of London, Royal Holloway, UK

ABSTRACT

After making the decision to use public-key cryptography, an organisation still has to make many important decisions before a practical system can be implemented. One of the more difficult challenges is to decide the length of the keys which are to be used within the system: longer keys provide more security but mean that the cryptographic operation will take more time to complete. The most common solution is to take advice from information security standards. This article will investigate the methodology that is used produce these standards and their meaning for an organisation who wishes to implement public-key cryptography.

1. Introduction

The power of public-key cryptography is undeniable. It is astounding in its simplicity and its ability to provide solutions to many seemingly insurmountable organisational problems. However, the use of public-key cryptography in practice is rarely as simple as the concept first appears. First one has to choose the type of public-key cryptography that is appropriate to solve the organisational problem in question. The array of choice is bewildering: public-key encryption schemes, digital signature schemes, signcryption schemes, identification schemes, attestation schemes, and any number of more specialised schemes with different functional properties and security guarantees.

After which the organisation has to develop an infrastructure to generate and distribute authentic copies of the public and private keys. The problems with developing an appropriate public-key infrastructure (PKI) have been widely documented. Even the importance of, and problems with, generating random keys have been widely discussed in both the practical and theoretical literature. One aspect of key generation has been largely ignored in the practical literature: the problem of choosing appropriate key lengths.

In a well-designed cryptographic scheme, a longer key means higher security (and a reduced chance of the system being compromised by an attacker). It also typically means a slower scheme. Most symmetric cryptographic schemes do not allow the use of keys of different lengths. If a designer wishes to offer a symmetric scheme which provides different security levels depending on the key size, then the designer has to construct distinct variants of a central design which make use of different pre-specified key lengths. (In other words, while AES-128 and AES-256 are based on the same principles, they have different internal components which have been chosen to complement the different key lengths). In particular, the allowable key lengths are chosen by the scheme designer. Public-key schemes are different – they typically have one design which can be used with keys of any length (without further input from the scheme designer). Hence, it is up to the user, rather than the designer, to determine an appropriate key length for use with the scheme.

This often leaves a complicated mathematical decision in the hands of a mathematically inexperienced security manager. The most common mechanism for determining an appropriate key length is to consult a standard. There are several standards which give key length recommendations for common public-key technologies (e.g., NIST, 2007; ECRYPT-II, 2010). To aid security managers in making sensible decisions, these standards typically give key lengths in terms of the amount of time that the key will "work" – i.e., the amount of

E-mail address: A.Dent@rhul.ac.uk.

1363-4127/$ – see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.istr.2010.10.006

time that data secured by a key of this length can be considered secure. The mathematical source of the numbers in these standards, and their practical meaning, is often misunderstood. This article will investigate the methodologies which produce these standards and attempt to give some intuition as to whether the advice they provide is suitable for a particular cryptographic scheme.

2. Key-length standards

For the purposes of this article we will differentiate between a cryptographic primitive and a cryptographic scheme. A cryptographic scheme is a set of practical algorithms that can be used within an organisation to solve some security problem. A cryptographic primitive is a basic piece of mathematics on which cryptography can be based, but which is not suitable for use as a cryptographic scheme on its own. For example, the RSA transform (Rivest et al., 1978) is a primitive, as it can be used to build cryptographic scheme but is generally not considered practical for use as a cryptographic scheme on its own. There are a bewildering array of different public-key schemes, but the vast majority of practical schemes are based on a few basic primitives:

- *RSA- or factoring-based*. For example, RSA-OAEP encryption, RSA-KEM encryption, HIME(R) encryption, RSA full domain hash signatures, RSA-PSS signatures, Rabin signatures, etc.
- *Discrete-logarithm-based*. For example, DHIES encryption, PSEC encryption, DSA signatures, ElGamal signatures, Schnorr signatures, HMQV key agreement, etc.
- *Elliptic-curve-based*. For example, ECIES encryption, PSEC encryption, ECDSA signatures, etc.

Therefore, the first step in determining secure parameters for a cryptographic scheme is to determine secure parameters for the underlying primitive. This is a highly technical mathematical task and there are a number of important research papers which attempt to answer this question (Blaze et al., 1996; Lenstra and Verheul, 2001; Lenstra, 2004). The conclusions of these research papers have been presented as guidance to the security community in the form of international standards or recommendations (e.g., NIST, 2007; ECRYPT-II, 2010). These standards typically recommend key sizes by comparing them to symmetric keys with equivalent security and/or by giving an estimate for the length of time that the key will keep data secure. We give a some examples of the key size recommendations[1] in Tables 1 and 2.

It is important to understand the meaning of the validity period. The validity period tells you how long data that has been protected with a key of a certain length can be considered secure – i.e., after the expiry date, data secured by a key can no longer be considered secure. It is not meant to serve as a guide for when a key should replaced or updated. There are good reasons to change keys before the expiry date, for example because it reduces the impact of key exposure. The use of long keys should not be thought of as a substitute for good key management and key lifecycle practices.

[1] See also http://www.keylength.com/.

Table 1 – ECRYPT-II key size recommendations (ECRYPT-II, 2010).

Equivalent symmetric key size	Validity	RSA-based modulus size	Discrete-log-based		Elliptic-curve-based key size
			Key size	Group size	
80	Up to 2014	1248	160	1248	160
112	Up to 2020	2423	224	2423	224
128	Up to 2040	3248	256	3248	256
160	–	5312	320	5312	320
192	–	7936	384	7936	384
256	Foreseeable future	15424	512	15424	512

It is also important to understand the source of these numbers. For certain symmetric key schemes, it is possible to prove that no attacker can break the scheme, regardless of the amount of time or computational resources they apply to the problem, e.g., Shannon's one-time pad encryption scheme (Shannon, 1949). This can never be the case for public-key cryptography: given a public key, and sufficient time, an attacker can always deduce the corresponding private key and thus break the scheme. The best we could ever hope to achieve is to show that an attacker is unlikely to break the scheme in a given period of time. However, even with this more modest goal, the mathematics is unfavourable. A proof that there exists no strategy which would break a public-key scheme with any reasonable probability would answer a famous open problem in theoretical computer science – the NP $\neq$ P conjecture – for which the Clay Mathematics Institute[2] have offered a \$1M prize.

So, if there exists no mathematical way to bound the probability that an attacker can break a scheme within a certain time, how do the standards generate reliable recommendations for key sizes? Not all standards bodies chose to release their methodologies for assessing key-length security; however, the publicly-known methodologies seem very similar (Lenstra and Verheul, 2001; Lenstra, 2004; ECRYPT-II, 2010). The key principle is that, since we cannot mathematically prove a bound on the probability of an algorithm breaking a scheme, we rely on the opinion of skilled mathematicians who understand the algorithms that may be used to break the underlying mathematical primitives, assessed through a three stage process.

In the first stage, the mathematicians attempt to find a relationship between the time it would take the best-known algorithms to break a random example of the underlying primitive and the time it would take an algorithm to break a symmetric key scheme through exhaustive search using technology available at a particular point in time. For example, in the case of breaking the RSA primitive, we examine the best-known algorithms for inverting a randomly-chosen ciphertext produced by encrypting a randomly-chosen message using a randomly-chosen RSA public key. The best-known algorithm to break the RSA primitive involves factoring the RSA modulus. We know from historical data that

[2] http://www.claymath.org/millennium/.

Table 2 – NIST key size recommendations (NIST, 2007).

Equivalent symmetric key size	Validity	RSA-based modulus size	Discrete-log-based Key size	Discrete-log-based Group size	Elliptic-curve-based key size
80	Up to 2010	1024	160	1024	160
112	Up to 2030	2048	224	2048	224
128	Beyond 2030	3072	256	3072	256
192	–	7680	384	7680	384
256	–	15360	512	15360	512

factoring a 512-bit RSA modulus was roughly equivalent to performing an exhaustive search on a 50-bit symmetric key in 1999. This starting point can used to extrapolate a formula that relates different RSA key lengths to their equivalent symmetric key lengths in 1999.

We equate public-key lengths with their equivalent symmetric-key lengths because it is much easier to estimate the amount of time for which a symmetric key will provide security (especially as we are only concerned with exhaustive search attacks). It is easy to estimate the number of computer commands it would take to break a symmetric key of a particular length. This gives rise to an estimate for the amount of time it would take to break a public-key primitive using technology available at a particular time (e.g., in 1999). The researchers then use a mathematical model which relates the time it would take to break a public-key primitive using historical technology to the amount of time that it would take to break that primitive using current technology. This model may include factors such as (Lenstra and Verheul, 2001; Lenstra, 2004):

- *Moore's Law*. This models the increase in computing power that can be obtained for the same cost over time. Roughly speaking, Moore's Law states that computing power doubles every 18 months. Thus, the amount of time taken to break the primitive might roughly be expected to halve over an eighteen month period.
- *Budget Increases*. This models the increase in budget that may be available to the attacker over time. For example, we may assume that the attacker's budget doubles every two years. If an algorithm can be efficiently parallelised, then we might expect the amount of time taken to break the primitive might halve every two years.
- *Cryptanalytic Advances*. This models the advances in cryptography which give rise to better attack strategies against the public-key primitives. Obviously, there can be no cryptographic advances that increase the effectiveness of a brute force search against a symmetric key. However, there may well be advances in cryptography which improve the algorithms for breaking the public-key primitive and so renders the equivalence deduced in the first stage of the analysis obsolete. If we were to assume that cryptographic advances mean that the time taken to break a public-key primitive halves every two years, then we can compensate for this by assuming that the time taken to find a symmetric key by brute force also halves every two years.

This gives rise to an estimate for the amount of time it would take for an attacker to break a primitive in any particular year.

Lastly, the researchers decide what is an acceptable length of time to give a reasonable security margin. In other words, we have to decide the security level which is acceptable for different applications. A key which is meant to be used with high-security applications may need to be longer than a key which is meant to be used for other applications. This is, again, a judgement that has to be made by experienced security experts.

There are several interesting points to be noted about this process. The first is that it is only concerned with security of the underlying primitive. The second is that it does not give an estimate of the amount of time t' required by an attack strategy to have a probability of ε' of breaking a primitive – it extrapolates the amount of time required to break the scheme based on previously successful strategies and mathematical models of how successful these strategies would be for longer keys or in later years. As we shall see, both of these points have a subtle impact when considering the security of a key length for a cryptographic scheme.

3. Provable security and concrete security

The estimates that we have for secure key lengths for primitives seem to be, for all practical purposes, reliable. Is this the end of the key length story? Sadly not. Consider the following nonsensical argument:

If the RSA primitive was insecure then all RSA-based schemes would be insecure too.

For sufficiently large key lengths, the RSA primitive is secure.

Therefore, all RSA-based schemes are secure.

This argument has the same structure as:

If the author was not a man then the author could not be King of England.

The author is a man.

Therefore, the author is King of England.

The argument does not take into account the fact that there might be other reasons why the author is not King of England despite the fact that his gender is consistent with the job requirement. Similarly, the original argument does not take into account the fact that there might be reasons why a cryptographic scheme is insecure despite the fact that the underlying primitive is secure.

This is not just a theoretical issue – the EMV payment system uses a digital signature scheme based on the RSA primitive (EMV, 2008). At a conference in 2009, a team of cryptographers announced an attack against the EMV signature scheme and implemented the attack against the scheme with a 2048-bit RSA modulus (Coron et al., 2009). This scheme had been widely considered to be secure and the attack was implemented against a key which the NIST recommendations claim should be secure until 2030. The attack did not break the RSA primitive but broke the way in which the RSA primitive was used to create a practical signature scheme. (It should be noted that the attack against this signature scheme does not imply an attack against the EMV payment system, which remains secure despite the fact that the signature scheme is flawed, although this attack may have serious repercussions for other systems that make use of the EMV signature scheme).

If we think of the methodology used to estimate secure key lengths then it quickly becomes apparent as to why a secure mathematical primitive does not immediately give a secure cryptographic scheme. As we discussed in Section 2, secure key sizes are estimated by examining the length of time that the best-known algorithm would take to break a random instance of the primitive. So, for example, when estimating secure key sizes for the RSA primitive, we investigate the time that the best-known algorithm would take to decrypt a randomly-chosen ciphertext for a randomly-chosen public key. However, in the real would, ciphertexts are not randomly-chosen – they are encryptions of real-world messages. Furthermore, in the real-world, an attacker isn't simply trying to invert a given ciphertext, but is usually trying to break into an interactive computer system, which gives the attacker lots of new ways to break the encryption. If you look at it in this light, it is amazing that any security primitive gives rise to a secure scheme at all!

The branch of cryptography which tries to relate the security of a scheme in a real-world situation to the security of the underlying mathematical primitive is called provable security. If we can present a mathematical proof that the difficulty of breaking a scheme is as hard as the difficulty of breaking some underlying mathematical primitive then we say that the scheme is provably secure or that it has a proof of security. The term "provable security" is something of a misnomer: a proof of security does not actually prove that the scheme is secure! In order to prove that a scheme is secure we would have to show that the underlying primitive cannot be broken, which, as we discussed in Section 2, is unlikely to happen in the near future. A proof of security demonstrates that any attack strategy which could break the scheme gives rise to an algorithm which breaks the underlying primitive. If we assume that the underlying primitive is secure, based on empirical evidence, then we can conclude that the scheme must also be secure. In this situation, we sometimes say that the problem of breaking the scheme reduces to the problem of breaking the underlying mathematical primitive and some authors have tried to replace the term provable security with the term reductionist security in order to remove the connotation of an absolute security guarantee given by the traditional term.

The provable security methodology works in two stages: (1) one defines a mathematical model which describes the way in which the attacker can interact with the cryptographic scheme and what it means to say that an attacker has broken the scheme, and (2) one shows that, in this model, an attack strategy that breaks the scheme can be adapted to give an algorithm which breaks the underlying mathematical primitive. This leads to lots of statements of the form,

If there exists an attack strategy against the cryptographic scheme that runs in time at most t and succeeds in breaking the scheme with probability at least ε when using keys of length ℓ, then there exists an algorithm which runs in time at most t' and succeeds in breaking the underlying mathematical primitive with probability at least ε' (where t' and ε' can be computed from t, ε and ℓ).

This gives a viable strategy for choosing key sizes for a particular cryptographic scheme. First, we choose the level of security that we require, by choosing values for t and ε which meet our security needs (i.e., that we would like there to exist no attack strategy that runs in time t which has success probability at least ε). Then we can increase the value of ℓ until the proof of security tells us that such an attack strategy would give rise to an algorithm which breaks the underlying primitive faster than the best-known current algorithms[3] If we assume that such an algorithm does not exist, then we can conclude that a key of length ℓ must be secure. This approach is known as concrete security.

4. Criticisms of concrete security

The use of concrete security, and provable security in general, has been criticised in a series of articles by Koblitz and Menezes (Koblitz and Menezes, 2006; Koblitz and Menezes, 2007; Koblitz, 2007). We will examine their, and other, criticisms in this section.

4.1. *"Provable security"*

As we discussed in Section 3, the field of "provable security" is somewhat misnamed, as a "proof of security" doesn't actually give a proof that a scheme is secure. A correct "proof of security" only assures you that for, long enough public keys, the scheme is as secure as the underlying primitive. Since we cannot prove that any of the underlying primitives are secure, our "provable security" results must necessarily be conditional. Unfortunately, the time has passed where this misnomer might be corrected and the term "provable security" is unlikely to change in the near future.

4.2. *Trust in the primitive*

A proof of security only guarantees security if the underlying primitive is secure. The majority of the schemes used in practice are based on the RSA problem, the discrete logarithm problem, or the elliptic-curve discrete logarithm problem. These primitives have been widely studied and are trusted to be secure. Furthermore, mathematicians have had decades of experience in constructing algorithms which attempt to break these primitives and, as we discussed in Section 2, the only way we can assess the security of a primitive is to estimate the time that the best-known algorithm would take to break the primitive. We trust these underlying primitives because of the time that the experts have taken in studying them.

However, over the last ten years, the cryptographic community has developed new technologies and new primitives on which to base cryptography, such as elliptic-curve pairings and mathematical lattices. These new technologies have allowed the cryptographic community to develop new types of cryptographic scheme with functional properties and

[3] At least, as far as we are able to tell – remember that the key-length standards do not attempt to give a bound on the probability that an attack strategy running in time t' has probability ε' in breaking the primitive.

security guarantees which were not possible with the original triptych of primitives. Provable security allows us to relate the security of these new schemes to well-defined underlying primitives, but it does not tell us the strength of the underlying primitive. It is hard to accurately estimate key lengths for these new primitives using the methodology in Section 2, and so it is hard to estimate practical key lengths which give rise to secure and efficient systems.

4.3. *The credibility problem*

Another unexpected problems comes from the credibility of individual security proofs. A security proof is not a calculation, but a complex mathematical argument that often comprises many pages of technical statements. It is easy for one or more mistakes to creep into these arguments. Security proofs can only be trusted if they have been thoroughly checked by a range of experts and deemed to be correct. The majority of standardised schemes have security proofs which have been checked and re-checked many times; thus, we can consider them to be correct. However, cryptographic schemes which are newer, or are based on more obscure principles, may not have had a sufficient level of scrutiny to determine if the security proof is correct or not.

4.4. *Moving outside the model*

As discussed in Section 3, a security proof only considers the security of a scheme in a mathematical model of reality. This problem is endemic: it is impossible to make mathematical statements except within a mathematical model and it is impossible to capture the complexities of the real-world in a mathematical model! Inevitably this means that there may be ways to attack a "provably secure" cryptographic scheme which lie outside the mathematical model of reality in which the scheme has been proven.

For example, the majority of proofs of security assume that the cryptographic scheme is an algorithm running on a "perfect" computer, which takes inputs and gives outputs without revealing any other information or making mistakes. However, in reality, computer chips can be unreliable and often leak information about the data that they are processing through real-world emanations. Examples of these "side channels" include the amount of time taken to perform a computation, the amount of power consumed while performing a computation, and the form of the electro-magnetic radiation given off during the computation. All of these side channels can be used to break the security of cryptographic schemes in practical situations and there has been little satisfactory progress on designing security models which take these types of attack into account.

Thus, a security proof only provides evidence as to the security of a cryptographic scheme if the scheme is deployed in an real-world environment which is consistent with the mathematical model of reality described in the security proof. The construction of meaningful models of reality is a major research challenge for cryptographers even when one doesn't consider troublesome issues such as physical side channels. In theory, each usage scenario for a cryptographic scheme should prompt a new model of reality; however, the cryptography community typically attempts to construct schemes which are secure in some kind of "supermodel" which guarantees security in any practical model of reality in which the scheme could be used. For the majority of the different types of public-key scheme, these models are now fixed, but even within these models mistakes can be made. The attacks of (Albrecht et al., 2009) against the (provably secure) SSH protocol (Bellare et al., 2004) are only feasible because the model in which the SSH protocol was proven secure does not match the way in which the scheme is used in practice.

4.5. *Tightness*

The previous sections discuss general problems with the use of provable security, rather than specific problems related to the difficulty of determining effective key lengths. However, there is one pivotal criticism of provable security that has a significant effect when examining key lengths – the problem of interpreting the tightness of the security proof.

Recall that the concrete security methodology, described in Section 3, uses the security proof to relate an attacker that runs in time t and has probability ε of breaking the scheme to an algorithm that runs in time t' and has probability at most ε' in breaking the underlying primitive. In early examples of security proofs, we often had that $t' \approx t$ and $\varepsilon' \approx \varepsilon$. We call such reductions "tight". A tight reduction can be interpreted (roughly) as saying that the security of the scheme is identical to that of the underlying primitive. For such schemes, the key-length standards can be used directly as guidance on the security of different key lengths.

However, as the complexity of security proofs has increased, the relationships between t, t', ε and ε' has become more complicated. How should we interpret a security proof[4] in which $t' \approx 2t$ and $\varepsilon' \approx \varepsilon^2$? Or a security proof[5] in which $t' \approx t + O(\varepsilon^{-2}\log(\varepsilon^{-1})\lambda^{-1}\log(\lambda^{-1}))$ and $\varepsilon' \approx \lambda\varepsilon/4$ for $\lambda = 1/8(n+1)q$, where n is a parameter of the scheme and q is a parameter of the allowable attack strategies? We call these reductions "loose" or "non-tight". If we apply the concrete security methodology to loose reductions then we will often find that we require keys to be unfeasibly large. Alternatively, if we use keys of the length recommended in the key-length standards, then loose proofs of security do not give any reasonable practical security guarantees.

Some authors, most notably (Koblitz and Menezes, 2006; Koblitz and Menezes, 2007; Koblitz, 2007), have argued that this means that loose security proofs have no practical meaning and should be ignored. Indeed, some authors have argued that loose security proofs are a waste of everyone's time and should not be considered a legitimate area of cryptographic study. This article does not take that position: we argue that all security proofs, even security proofs with loose reductions, are useful in determining practical key lengths.

[4] These are essentially the parameters found when using the forking lemma (Pointcheval and Stern, 1996) to produce proofs of security for signature schemes.

[5] These parameters are found in a security proof of (Waters, 2005), although a tighter security proof has now been found for the scheme in question.

Consider a security proof in which $t' \approx t$ and $\varepsilon' \ll \varepsilon$. It is tempting to believe that this should be interpreted as saying that there exists an attacker who can break the scheme in time t with success probability $\varepsilon \gg \varepsilon'$ and that therefore long keys are required. This is a fallacious conclusion. The security proof only says that if there did exist an attacker who can break the scheme with probability ε in time t then there would exist an algorithm which can break the underlying primitive with probability at least ε' in a similar time. It may be that an attacker that can break the scheme with probability ε always gives rise to an algorithm which can break the primitive with probability ε; the proof neither excludes this possibility nor guarantees it. In many cases, an examination of the mathematics of the security proof itself can give empirical evidence as to whether $\varepsilon \gg \varepsilon'$ in practice or not.

An examination of security proofs with loose reductions suggests that most of the time the "looseness" in the security reduction is not the result of a viable attack strategy against the scheme, but of the technical mathematics required by the proof strategy used to relate t, t', ε and ε'. As an example, we consider a common problem in security proofs related to the number of users of a scheme. In many security reductions, the proof of security shows that $t' \approx t$ and $\varepsilon' \approx \varepsilon/(\#users)$. The logic behind these reductions is simple: when assessing the security of the underlying primitive, we consider only a single instance of the primitive, which typically corresponds to a practical situation in which the attacker is trying to break a specific user's public key. However, in practice, the attacker may be satisfied with breaking anyone's public key and so his chance of success increases by a factor of *#users*. This seems logical: after all, there are attack strategies whose probability of success increases by a factor of *#users* when moving from attacking a single key to attacking multiple keys, for example a brute force strategy which simply generates public/private key pairs at random until it finds that it has generated some user's public key. Therefore, if we naïvely use the concrete security methodology, then we should increase key lengths depending on the number of users in the system, a quantity that seems to be entirely independent of the security level of the cryptographic scheme.

The problem is that there is a mismatch between the methodology used to estimate practical key lengths (Section 2) and the concrete security methodology (Section 3). The concrete security methodology has to give a relationship between t, t', ε and ε' for any attack strategy, even inefficient strategies such as brute force search; the methodology used to estimate practical key lengths only considered the best-known attack strategies. Most cryptographers would not recommend significantly increasing individual users' key sizes in systems where there are a large number of users, despite the fact that the security proof may recommend an increase related to *#users*, because the best-known attack strategies can only target one key at a time. Since the best-known attack strategies can't be used to attack *#users* keys simultaneously, there is no need to take notice of this term in the security proof when considering practical key lengths.

However, there are also examples of situations where a security proof has shown a practically significant gap between the security of the scheme and the security of the underlying primitive. For example, a theoretical analysis of a new cryptographic primitive, called the d-strong Diffie-Hellman problem, suggested that it might be weaker than other similar problems by a factor of $\sqrt{d}$ (Boneh et al., 2004). This was later confirmed through a valid attack strategy (Cheon, 2006). A thorough understanding of the algorithms that can be used to attack a primitive and the way in which the security proof relates the security of the scheme to the security of the primitive can highlight differences between the primitive and the scheme which may result in the scheme requiring longer keys to be practically secure. Thus, we argue that even loose security reductions can be very useful in determining appropriate key lengths, when used as evidence by a qualified mathematician who can interpret their practical significance correctly.

5. Conclusion

For an organisation who wishes to use public-key cryptography to protect high-security assets, the difficulties in choosing sensible key lengths are clear. We may estimate the security of the underlying primitive, but our estimates can never be completed trusted and do not necessarily tell us the security of the actual cryptographic scheme. Provable security results can be used to provide some evidence as to security of the scheme, but security proofs must be interpreted correctly if they are to be used effectively. Almost all standardised schemes now have a proof of security, but these proofs often run to a dozen pages when presented in their entirety. It can easily be argued that this gives an overload of information, which actually inhibits sensible decision-making strategies. In order to effectively make use of all the information provided by a security proof, an organisation needs two different sorts of cross-discipline experts:

- An expert in provable security and the mathematical algorithms that can be used to break cryptographic primitives (who can make a sensible assessment of the security reduction and its implications for the choice of an appropriate key length).
- An expert in mathematical models and secure implementations (who can make a sensible assessment of the extra measures required to prevent attacks which are not included in the mathematical model of reality used by the security proof).

It is unclear if even the most security conscious of organisations will be able to justify the time and expense of employing these experts to accurately estimate the length of secure and efficient keys.

Luckily, we may always turn to standardisation bodies to help amortise the cost of these experts over many organisations. The existing key-length standards (e.g., NIST, 2007; ECRYPT-II, 2010) provide reliable estimates for the security of the basic primitives, but these cannot be completely trusted to provide estimates of the security of the cryptographic schemes based on these primitives. Standardisation bodies are slowly beginning to recognise this fact and there are some attempts to produce application-specific key length guidelines, e.g., (NIST, 2009). Furthermore, as the next generation of cryptographic schemes are being developed, more effort is needed to determine secure and efficient parameter sizes for the new primitives that these schemes use.

REFERENCES

Albrecht MR, Paterson KG, Watson GJ. Plaintext recovery attacks against SSH. In: IEEE Symposium on Security and Privacy. IEEE Computer Society; 2009. p. 16–26.

Bellare M, Kohno T, Namprempre C. Breaking and provable repairing the SSH authenticated encryption scheme: a case study of the Encode-then-Encrypt-and-MAC paradigm. ACM Transactions on Information and System Security 2004;7(2):206–41.

M. Blaze, W. Diffie, R.L. Rivest, B. Schneier, T. Shimomura, E. Thompson, and W. Wiener. Minimal key lengths for symmetric ciphers to provide adequate commercial security. Manuscript, January 1996.

Boneh D, Boyen X, Shacham H. Short group signatures. In: Franklin M, editor. Advances in Cryptology – Crypto 2004. Lecture Notes in Computer Science, vol. 3152. Springer-Verlag; 2004. p. 41–55.

Cheon JH. Security analysis of the strong Diffie-Hellman problem. In: Vaudenay S, editor. Advances in Cryptology – Eurocrypt 2006. Lecture Notes in Computer Science, vol. 4004. Springer-Verlag; 2006. p. 1–11.

Coron J-S, Naccache D, Tibouchi M, Weinmann R-P. Practical cryptanalysis of ISO/IEC 9796-2 and EMV signatures. In: Halevi S, editor. Advances in Cryptology – Crypto 2009. Lecture Notes in Computer Science, vol. 5677. Springer-Verlag; 2009. p. 428–44.

EMVCo. EMV Integrated circuit card specifications for payment systems: Book 2 — Security and key management; June 2008.

European Network of Excellence in Cryptology II. ECRYPT II yearly report on algorithms and keysizes; March 2010.

Koblitz N. The uneasy relationship between mathematics and cryptography. Notices of the AMS 2007;54(8):972–9.

Koblitz N, Menezes AJ. Another look at "provable security" II. In: Barua R, Lange T, editors. Progress in Cryptology – INDOCRYPT 2006. Lecture Notes in Computer Science, vol. 4329. Springer-Verlag; 2006. p. 148–75.

Koblitz N, Menezes AJ. Another look at "provable security". Journal of Cryptology 2007;20(1):3–37.

Lenstra AK. Key lengths. In: Bidgoli H, editor. Handbook of information security. John Wiley & Sons; 2004.

Lenstra AK, Verheul ER. Selecting cryptographic key sizes. Journal of Cryptology 2001;14(9):255–93.

National Institute of Standards and Technology (NIST). NIST SP800–57: Recommendation for key management – Part 1: general; March 2007.

National Institute of Standards and Technology (NIST). NIST SP800–57: Recommendation for key management – Part 3: application-specific key management guidance; Dec 2009.

Pointcheval D, Stern J. Security proofs for signature schemes. In: Maurer U, editor. Advance in Cryptology – Eurocrypt '96. Lecture Notes in Computer Science, vol. 1070. Springer-Verlag; 1996. p. 387–98.

Rivest RL, Shamir A, Adleman L. A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM 1978;21:120–6.

Shannon CE. Communication theory of secrecy systems. Bell System Technical Journal 1949;28:656–715.

Waters B. Efficient identity-based encryption without random oracles. In: Cramer R, editor. Advances in Cryptology – EUROCRYPT 2005. Lecture Notes in Computer Science, vol. 3494. Springer-Verlag; 2005. p. 114–27.