The three pillars of cryptography

version 1, October 6, 2008

Arjen K. Lenstra

EPFL IC LACAL
INJ 330 (Bâtiment INJ), Station 14
CH-1015 Lausanne, Switzerland

Abstract. Several aspects are discussed of 'Suite B Cryptography', a new cryptographic standard that is supposed to be adopted in the year 2010. In particular the need for flexibility is scrutinized in the light of recent cryptanalytic developments.

1 Introduction

Securing information is an old problem. Traditionally it concerned a select few, these days it concerns almost everyone. The range of issues to be addressed to solve current information protection problems is too wide for anyone to grasp. Sub-areas are so different to be totally disjoint, but nevertheless impact each other. This complicates finding solutions that work and is illustrated by the state of affairs of 'security' on the Internet.

In the early days of the Internet the focus was on technical solutions. These still provide the security backbone, but are now complemented by 'softer' considerations. Economists are aligning security incentives, information security risks are being modeled, legislators try to deal with a borderless reality, users are still clueless, and psychologists point out why. According to some we are making progress.

To improve the overall effectiveness of information protection methods, it can be argued that it does not make sense to strengthen the cryptographic primitives underlying the security backbone. Instead, the protocols they are integrated in need to be improved, and progress needs to be made on the above wide range of societal issues with an emphasis on sustainably secure user interfaces (cf. [3]). But, assuming the situation improves on those fronts, we also have to make sure to stay abreast of cryptanalytic advances that may undermine the cryptographic strength.

In this write-up a development is discussed that is triggered by 'Moore's law': roughly speaking the effective cost, i.e., taking inflation into account, of computing drops 100-fold every decade. This affects the security of cryptographic standards, since they become a million times easier to break over a 30 year period. So, irrespective of the actual security offered by a system as a whole, the strength of the underlying cryptographic standards needs to be constantly monitored and occasionally the standards need to be upgraded. We are now on the brink of such an upgrade. Although we believe that this upgrade is timely and appropriate, there are several comments that can be made. Those comments are the subject of this write-up.

The upgrade in question is *Suite B Cryptography* (cf. [13] and [14]). Assuming continuation of Moore's law for the decades to come and no cryptanalytic disturbances, replacement of current cryptographic standards by Suite B Cryptography would extend the life span of our cryptographic primitives by more than 30 years. That would provide cryptographic security beyond anyone's imagination – and most certainly completely out of sync with the true security that can realistically be achieved when Suite B is integrated in current systems. Nevertheless, it is a lofty goal.

The first point to be addressed is the assumption about the continuation of Moore's law. According to several sources (such as [5]) this assumption can indeed be made for another three or four decades. However, the cost decrease will no longer be achieved by greater processor speeds, but by increased parallelization. The extent to which computational tasks can benefit from higher degrees of parallelism varies tremendously. So far, cryptanalytic calculations are trivially and fully parallelizable, since they mostly involve stand-alone processes that can be run in parallel with and mostly independent of any number of similar processes. There is no reason to suspect that this will change. It is therefore reasonable to take the full potential effect of Moore's law into account in our estimates.

Suite B was presented by the United States National Security Agency (NSA), citing [14],

> to provide industry with a common set of cryptographic algorithms that they can use to create products that meet the needs of the widest range of US Government (USG) needs

and

> is intended to protect both classified and unclassified national security systems and information.

Driven by their need for 'Commercial Off the Shelf' products, the NSA coordinated with the United States National Institute of Standards and Technology (NIST) to standardize the algorithms in Suite B, stressing that there is no difference between the algorithms for 'Sensitive But Unclassified' (SBU) and classified use. NIST extended Suite B with a few primitives for SBU use, and recommends adoption of the new standard by the year 2010 (cf. [13]).

Although it is a US standard, it is likely to have worldwide impact. Both from an economic and logistic point of view, it is hard to imagine that manufacturers or vendors would adopt products that would be excluded from a part of the US market or that large corporations would switch systems between continents. Furthermore, it remains to be seen if there will be demand for products based on a wider range of primitives than included in Suite B.

Usage of a well defined set of cryptographic primitives reduces diversity of implementations and incompatibilities and will, overall, facilitate communications. Thus, adoption of a worldwide standard could be beneficial. However, before doing so everyone involved should be aware of the full range of potential implications. We are not in a position to provide any type of feedback on a US standard that resulted from a collaboration between NIST and NSA. Nevertheless, we recommend that all parties considering adoption of Suite B form their own opinion before committing to it.

The opening sentence of [14] reads, promisingly,

> The sustained and rapid advance of information technology in the 21st century dictates the adoption of a flexible and adaptable cryptographic strategy for protecting national security information.

Although it is not explicitly stated, we assume that this is meant to imply that Suite B is both flexible and adaptable. We interpret 'flexible' as a property of Suite B itself, namely that it contains a sufficiently large variety of methods with similar functionalities to allow for immediate migration from a possibly compromised method to one that is believed to be secure. Thus, an implementation that contains the full Suite B is necessarily flexible. An example would be migration from one symmetric key size to the next. 'Adaptability', on the other hand, we interpret as a property of the implementation, namely that any of the main building blocks can relatively easily be replaced by a better one. Replacement of a current cryptographic hash function by the (as yet unknown) result of the recently launched NIST hash competition (cf. [12]) would be an example. An implementation of Suite B may be complete without being adaptable.

In this write-up we present a high level view of Suite B and its NIST extension, and comment on both. Our purpose is to raise the level of awareness about this to-be-adopted cryptographic standard. In Section 2 we discuss the security of cryptographic primitives. A summary of the new standards is presented in Section 3. More details concerning Suite B and block ciphers, cryptographic hashing, and public-key cryptography are discussed in sections 4, 5, and 6, respectively.

2 Background

The security of cryptographic primitives is measured in bits: k-bit security means that on average 2^k basic computational steps are required for a successful attack. A 'basic computational step' depends on the primitive and the nature of the attack: it requires a number of machine instructions that is a relatively small constant or that grows as a small function of k. Moving, for a given primitive and attack, from k-bit to $(k+1)$-bit security roughly doubles the effort required for a successful attack.

Interpreting this in financial terms, attacking a fixed security level gets cheaper over time. Thus, if a certain minimal attack cost needs to be maintained over time, the security as measured in bits cannot remain constant. Its minimal rate of change must keep up with Moore's law: per decade the effective cost to mount a successful attack against a certain primitive at a fixed security level drops by a factor of one hundred. With

$$1.5 * 10 = 15 \text{ and } 100^{1.5} = 1000 \approx 2^{10} \tag{1}$$

we find that every fifteen years we need to add 10 bits to the security level if the same attack cost must be maintained. This does not take into account the unpredictable effect of improved attacks,

Traditional implementations of cryptographic primitives are not exactly adept at changing security levels. For some it requires 'just' larger keys, which sounds easy but is hard, if not impossible. For others it requires a design overhaul, which requires lots of time and money and – worse – consensus. In any case, changing security levels is costly and should be avoided unless the risk of increased exposure becomes unacceptable. Recognizing this fact, and in an attempt to have a 'buffer' against unforeseen (since unforeseeable) cryptanalytic progress, standards generally prescribe security levels corresponding to attacks that can for a sufficiently long period of time be expected to be prohibitively expensive relative to the value of the protected data. Despite this precaution, it is unavoidable that every now and then this security margin drops below acceptable levels, due to Moore's law, cryptanalytic progress, or a combination of both.

At this point in time, and for quite a while already, industrial standards for cryptographic primitives have been prescribing 80-bit security. The current cost of successfully attacking such primitives is considerable. On a processing unit that would be capable of performing 10^{10} 'basic computational steps' per second, an attack would require about 4 million years. Since virtually all attacks are fully parallelizable, this would be equivalent to saying that a successful attack would require a year on 4 million such units[1]. This is out of reach for the average miscreant, but not for the very wealthy or state-funded ones. Worse is that in ten years the security margin will be a hundred times smaller, which is too close for comfort even given our overly optimistic assumption concerning the processing unit's speed. This holds irrespective of the security offered by the 'rest' of the system: we should avoid a situation where attacking the cryptography would become the most attractive way to undermine a system's security. Combined with fairly recent cryptanalytic progress that chipped off about 20 bits of one of the 80-bit primitives, we conclude that we should adopt standards of a substantially higher level of security than 80 bits, while at the same time overhauling the design of the actually broken one. Suite B Cryptography intends to do the former, i.e., increasing the security level. This is set forth in the next sections. The required overhaul of the 'broken' cryptographic primitive is a parallel development (cf. [12]) the outcome of which is still uncertain.

[1] For at least one current 80-bit security standard this estimate is off by a factor one thousand: according to [1] a year on 4 billion machines is close to what one should expect.

3 Suite B Cryptography

Suite B Cryptography comes in two flavors. The NSA original [14] aims for 128-bit security or more and prescribes a single method for each functionality specified. The NIST version [13] is satisfied with 112-bit security and allows greater flexibility in the choice of methods. With $112 - 80 = 32$, a 112-bit cryptographic primitive is roughly 2^{32} times harder to break than an 80-bit one. Because 2^{32} is close to $(2^{10})^3 \approx 1000^3$ and 3 times 15 years is about half a century (using (1)), it takes about that long after the adoption of the new NIST standard before we find ourselves again in the situation that we need to reconsider the security level. The NSA's 128-bit Suite B would add another 25 years, for a total of about 70 years. These estimates assume Moore's law will hold the next 5 (or even 7) decades and no cryptanalytic breakthroughs. Both sound unlikely, for different reasons, so these figures should be taken with a large grain of salt. Nevertheless, as far as we can tell now, security-level-wise either choice is adequate.

Both Suite B and its NIST variant prescribe cryptographic primitives for four basic functionalities:

1. Symmetric encryption.
2. Cryptographic hashing.
3. Digital signatures.
4. Key exchange.

The last two functionalities are based on public-key cryptography and are therefore treated as a single category in this paper. The three resulting categories are what we refer to in the title of this write-up as the three pillars of cryptography. The next three sections discuss the three pillars in more detail.

4 Symmetric Encryption

Two parties that share a *key* can use *symmetric encryption* to protect their communication. One party uses the shared key to *encrypt* data and transmits the encrypted data to the other party. The other party uses the same key to *decrypt* the encrypted data, which should result in the original data. For anyone who does not know the shared key it should be computationally infeasible to interpret the transmitted data; ideally this should require searching the entire *key space*.

To agree on a key to use for symmetric encryption, the parties first have to engage in a *key agreement* protocol. This is discussed below in Section 6. Symmetric encryption is typically based on *stream ciphers* or on *block ciphers*. Stream ciphers are not included in Suite B or its NIST variant. Given the state of flux of stream cipher development and cryptanalysis, and despite the efforts of the ECRYPT stream cipher project *eSTREAM* (cf. [2]), inclusion of any currently still surviving stream cipher proposal would indeed have been premature. Thus, for symmetric encryption Suite B and its NIST variant both rely entirely on block ciphers.

A key for a block cipher is a sequence of bits of some fixed prescribed length denoted by ℓ. Typically, a key is chosen at random from the key space, which is either $\{0,1\}^\ell$ or a subset thereof. A *block* is a sequence of bits of some fixed pre-specified length denoted by b. Typically $b = 64$ or $b = 128$. Given a key, a block cipher treats one block at a time. Data of arbitrary length to be symmetrically encrypted using that key therefore first needs to be written as a sequence of blocks. How this is done, possibly including *padding*, and which *mode of operation* is used to process the resulting sequence by means of the block cipher, is not described in detail in Suite B, but a link is provided. These issues are also beyond the scope of the present write-up.

For any key in the key space $\mathcal{K}$, a block cipher C is a bijection on the space of *data blocks* $\{0,1\}^b$:

$$C : (\mathcal{K}, \{0,1\}^b) \to \{0,1\}^b.$$

So, for any key $K \in \mathcal{K}$ and any data block $B \in \{0,1\}^b$

$$C^{-1}(K, C(K, B)) = B.$$

Both C and its inverse must be efficiently computable. In practice this means one may expect to spend a small constant number (generally at most 5, and rarely more than 10) of machine cycles per bit to be encrypted or decrypted using a software implementation.

The NSA original of Suite B prescribes the block cipher AES (Advanced Encryption Standard), a succinct description of which can be found in [11]. AES uses $b = 128$ and allows the full key space $\{0,1\}^\ell$ for $\ell \in \{128, 192, 256\}$. The NIST variant allows all three choices, implementation of NSA's original Suite B must include $\ell = 256$, excludes $\ell = 192$ 'for interoperability' (cf. [14]), and allows $\ell = 128$. Furthermore, the NIST variant also allows usage of 3DES (*triple DES*, where DES is the abolished Data Encryption Standard with key length 56) with $b = 64$ and $\ell = 3 * 56 = 168$.

Given these choices, the question is if they live up to the expectations: does AES with $\ell = 128$ indeed provide 128-bit security, does $\ell = 256$ indeed provide more, and do the NIST relaxations provide at least 112-bit security? As usual with questions of this sort, the answer depends on the attack model.

Restricting to traditional attack models, the answer is positive: according to the current cryptanalytic state of the art, ℓ-bit AES provides ℓ-bit security, and 3DES provides 112-bit security (note that 112 is quite a bit less than 3DES' key length of 168). Examples of such attack models are retrieving at least one of the plaintextblocks given just a number of ciphertextblocks (generated using the same key, say), or retrieving the key given a number of (plaintextblock, ciphertextblock) pairs.

There are more considerations, however. A first, relatively minor point is the following. Although 'everyone' knows that keys need to be occasionally refreshed, we have not been able to find in the relevant AES standard linked to from [14] an explicitly stated firm upper bound on the number of blocks that may be encrypted with the same key. This is slightly worrisome, as explained in the next paragraph.

Let S be a set of (plaintextblock, ciphertextblock) pairs, all generated with the same key. It is a well known fact that the chance that S contains the plaintext corresponding to a random ciphertextblock challenge (generated using the same key) grows much faster than linear as a function of $|S|$. This chance is already non-negligible for $|S| \approx 2^{b/2}$ (cf. birthday paradox), which does **not** depend on ℓ. Depending on how one counts the 'attack effort', it may be argued that AES with $\ell = 128$ (and $b = 128$) provides the desired 128-bit security. Namely, if one employs the 'time'$\times$'memory' cost model (cf. [23]) and one includes the time to generate S, then one is so fortunate to get attack effort $2^{b/2} \times 2^{b/2} = 2^b = 2^\ell$. For AES with $\ell > 128$ (but still with the unavoidable $b = 128$) or 3DES (with $b = 64$) there does not seem to be a way to argue that one gets the desired security level. Actually, in all these cases the gap is considerable: for instance, AES with $\ell = 192$ or 256 requires the same attack effort 2^{128} as does AES with $\ell = 128$, as argued above. An attack of this sort against AES requires half a billion Petabytes of storage, which is a lot but not undoable (for a fixed key a (plaintextblock, ciphertextblock) pair requires just $128 + 128 = 2^8$ bits, i.e., 2^5 bytes, of which about 2^{64} are needed, i.e., a total of 2^{69} bytes). For 3DES the storage requirement is just 2^{32+7} bits, i.e., 64 Gigabytes, which is very feasible.

Even with single-key encrypted Terabyte transmissions there is no serious trouble for AES. For 3DES, however, keys would need to be refreshed at least once every few Gigabytes, which may become cumbersome for large volume transmissions.

More worrisome is the following point. One of the selection criteria for AES was good performance both in software and hardware. Indeed, fast software and hardware implementations are available on a wide variety of platforms. To the best of our knowledge, the AES standard does not prescribe any specific restrictions on the environment or the circumstances under which any particular AES implementation must be used, but leaves this to the common sense of the users. Obviously, ‘others’ should not be able to access anyone else’s AES keys or be able to snoop around in their AES processes while they are running. Modern operating systems provide all the access control mechanisms to properly separate users from unauthorized others, with as perceived greatest dangers weak passwords and sticky pads. Thus, even though employees may have access to each others desktops, individual data and processes are neatly separated as long as all parties involved keep their access credentials to themselves.

The concept of *virtual machines* takes this a step further, allowing companies to economize on hardware by letting multiple employees share the same physical machine while giving each of them the look and feel of their own personal desktop computer. Anyone using such a work environment may, quite reasonably, expect that other users of the same physical hardware cannot access or learn anything for which they are not privileged. Unfortunately, with all the fancy access mechanisms and separation of processes and data, different users of the same piece of computing equipment, currently share at least one important resource, namely the *data cache.* When current CPUs switch between processes, the state of the cache persists, leading to inter-process contention for cache resources. As a consequence it is, fortunately, not the case that different users can read each others’ cached data, but users are able to observe parts of each others’ usage pattern of individual lines of the data cache: not the contents of a certain cache line can be accessed, but just the mere fact that a particular line of cache has been used can be observed.

The potential consequences of this leakage of meta-information vary from application to application. AES depends heavily on data dependent table lookups. Therefore, the consequences for AES of this type of *side channel attack* are particularly unfortunate, as shown in [15] for several different realistic scenarios. Roughly speaking, if a miscreant runs a certain cleverly designed process simultaneously with someone else’s AES process, the miscreant can retrieve the corresponding AES key in a negligible amount of time, with trivial computational effort, and in such a way that the owner of the key remains unaware of the attack – a far cry from the security level the unwitting key owner had every reason to expect. So far attempts have failed to produce efficient software implementations of AES that are not vulnerable to cache attacks. Another choice than Rijndael may have resulted in an Advanced Encryption Standard that resists cache attacks.

The threat of cache attacks is taken seriously by the relevant industries. Intel is the latest large processor manufacturer to provide hardware AES instructions (cf. [4]), following the lead of others (such as ADM and VIA) who have had this for a few years already. Given their novelty, these AES-on-chip implementations have not been fully scrutinized yet by the open community, and we are not aware of side-channel attacks affecting them.

Despite this promising development, AES may and will be used in software. But, as far as we know, no software usage restrictions or warnings have been specified in the standards, so uninformed parties may find themselves at risk. Obviously, no standard can anticipate all bad use cases and many important decisions will have to be left to the implementers and users. But problems that may occur in applications or environments that can be considered, from all reasonable points of view, as middle of the road, should be fully addressed.

5 Cryptographic hashing

A *cryptographic hash function* H of length h is an easily computable function that maps bitstrings of arbitrary length to bitstrings of length h:

$$H:\,\{0,1\}^* \to \{0,1\}^h.$$

Since h is fixed there exist $x, y \in \{0,1\}^*$ with $x \neq y$ and $H(x) = H(y)$. However, for a cryptographic hash function it is assumed to be computationally infeasible to generate such x and y, a property that is referred to as *collision resistance*. It is also assumed that H is *preimage resistant*, i.e., for any s in the range of H it is computationally infeasible to find an $x \in \{0,1\}^*$ such that $H(x) = s$.

Finally, it is assumed that H is *2nd preimage resistant*, i.e., for any $y \in \{0,1\}^*$ it is computationally infeasible to find an $x \in \{0,1\}^*$ such that $H(x) = H(y)$ and $x \neq y$. Note that in the last case existence of x can be expected but in general not be guaranteed.

The above assumptions are, more or less, the standard assumptions one will find in the literature about cryptographic hash functions. It is not clear if these are all properties that a cryptographic hash function should satisfy, but that would lead to a debate that is beyond the scope of this write-up. Concentrating on the more practical issues at hand, the computation speed of cryptographic hash functions is comparable to that of block ciphers, assuming inputs of similar lengths. As far as computational infeasibility is concerned, finding a preimage or a 2nd preimage should ideally require 2^h 'steps'. This requirement cannot be met for collision resistance because, due to the birthday paradox, one may expect that collisions can be found in $2^{h/2}$ steps. Any h-bit cryptographic hash function therefore has security at most $h/2$ as far as collision resistance is concerned. There exist cryptographic hash functions that have much longer output lengths than suggested by their resistance against preimage or collision attacks (i.e., 80-bit security against collision attacks, but output length 1024, which is much larger than the 160 one would 'expect'). Those functions are mostly of theoretical interest and will not be considered here.

It is not always clear when a cryptographic hash function should be considered broken. Even when collisions can be found in fewer than $2^{h/2}$ steps and (2nd) preimages can be found in fewer than 2^h steps, the latter problem may still be computationally infeasible by requiring many more than $2^{h/2}$ steps. Though applications that rely on preimage resistance are in principle affected, the protection should still be more than adequate. Nevertheless, even a 'partial break' of that sort is a sign of weakness of the design and a forewarning that should not be ignored. Furthermore, continued usage of a cryptographic hash function with just partial applicability is a recipe for disaster.

That the above is not a purely academic discussion is illustrated by recent events, which we briefly sketch. All common practical cryptographic hash functions are *iterative* and use the *Merkle-Damgård construction* with *strengthening*: roughly speaking, the input is padded with, among others, its length and partitioned into 512-bit *blocks*, which are processed by a *compression function.* It was proved that the resulting cryptographic hash function is collision resistant if the compression function is collision resistant. As it turned out, however, the iterative approach to cryptographic hashing also introduces undesirable weaknesses. The most infamous one is Antoine Joux's disturbingly simple observation that 2^k different hash collisions can be constructed based on k compression function collisions (cf. [6]). Reflecting the complacent attitude towards cryptographic hash function research, this was only discovered more than 15 years after the Merkle-Damgård construction was introduced. It immediately follows from Joux's observation that concatenating two cryptographic hash functions – a popular construction – does not result in a cryptographic hash function that has substantially stronger resistance against collision attacks than the weakest iterative one among its two constituents.

Matters were made considerably worse in the fall of 2004, when Xiaoyun Wang and her coworkers showed how certain differential properties of the compression functions of many popular cryptographic hash functions could be cryptanalytically exploited. In [19], [20], [21], and [22] a sequence of attacks is described, making it devastatingly easy to find collisions for some cryptographic hash functions that were already considered to be very weak (such as MD4), for the first time showing collisions for MD5 (still widely used, but at that time already known to be weak), finding faster collisions than before for SHA-0 (the replaced precursor of SHA-1, also known to be weak), and showing that finding collisions for SHA-1 is easier than it ought to be (though still challenging). Remarkably, some dismissed this work as irrelevant for practical applications, because all it achieved were collisions between more or less random looking values. Later generalizations and extensions made attack scenarios more realistic and threatening, a development that continues to the present day. At this point certification applications of MD5 should be discontinued, and if usage of SHA-1 can be avoided (such as in implementation of newly designed protocols) it is advisable to do so.

The cognoscenti immediately realized that something was seriously amiss when Xiaoyun Wang presented her results. After the exuberant cheers following her Crypto 2004 rump session presentation had died down, there was a clear sense of “now what?”. This can be explained by looking at the historical development of the currently most popular cryptographic hash functions and the demise of some of them. MD4 was an early 128-bit cryptographic hash function, quickly found to be unsatisfactory and strengthened, resulting in MD5 also of length 128. Although weaknesses emerged, MD5 gained widespread acceptance. To the present day MD5 is used in, among others, certification applications. In the early 1990s the *Secure Hash Algorithm* (SHA) was proposed, based on similar design principles as MD5, but of length 160. In 1994 SHA was unexpectedly withdrawn and replaced by SHA-1 which is identical to SHA with the exception of a single change which makes SHA-1 substantially more secure than SHA (due to a subtlety that the open community only fully understands since 1998). The original SHA is now commonly referred to as SHA-0. In 2002, an extension of the ideas behind SHA-1 led to the new cryptographic hash function SHA-2, more precisely to SHA-256 and SHA-512, of lengths 256 and 512, respectively. SHA-2 is the current state of the art. SHA-384 (essentially SHA-512 truncated to 384 bits) must be included in a Suite B implementation, and SHA-256 may be included as well. The NIST variant also allows SHA-512 and the 224-bit truncation SHA-224 of SHA-256.

Given the development of cryptographic hash functions sketched above, this exclusive reliance in the new standards on SHA-2 is understandable but, from an operational point of view, a bit odd. It is understandable because the basic design principle of SHA-2 has been extensively studied, the lighter weight earlier versions (up to and including SHA-1) are vulnerable, but SHA-2 seems to be invulnerable to those attacks. It is odd for several reasons. In 2005, the same invulnerability argument applied to SHA-1: after the MD4, MD5, and SHA-0 collisions had been announced and at a point that SHA-1 had not been affected yet, NIST declared its continued full support for SHA-1, only to be faced with an effective – and similar – cryptanalysis of SHA-1 a week later. Given the track record of the design principle used, it is surprising that it still enjoys such strong support. Another reason that exclusive adoption of SHA-2 in Suite B is odd is that, in recognition of potential problems with SHA-2, NIST has launched an open competition for a new cryptographic hash standard (cf. [12]). It would be appropriate to explicitly mention in Suite B, or at the very least in the NIST version of it, that a new standard is forthcoming, so that good citizens who adopt Suite B can prepare themselves for yet another change.

A third aspect of the unaltered choice of SHA-2 in Suite B is related to what the cryptographic grapevine has to say about the competition and its outcome, and to the nature of the weaknesses found so far. There seems to be fairly general agreement that the current understanding of cryptographic hash functions is not up to par with what we thought we understood about block ciphers at the start of the AES competition. If not at the design level (since, what do we *really* understand about block cipher design?), then most certainly at the requirement level: not even NIST knows what properties a cryptographic hash function should have (cf. [8]), and hopes that the pieces of that puzzle will fall into place as a result of the competition. This needs to happen quickly, as submissions are due in the fourth quarter of 2008. If a new standard will be adopted as a result of the competition, according to the current schedule by the year 2012, how much confidence can we have in a design that by then has been scrutinized for at most 4 years? How would that level of confidence compare to the confidence one may have in well studied approaches after known weaknesses have been addressed?

In an attempt to address this question, the main problem with current cryptographic hash standards seems to be twofold.

- **Number of rounds.** The iteration used in the compression function is not repeated sufficiently often to obviate exploitation of differential properties.
- **Message expansion.** The way new message bits are dealt with offers inadequate resistance against differential attacks.

The second point has in the mean time been studied. In [7] better message expansion methods are presented that entail only a small performance penalty and that allegedly offer resistance against differential attacks. This could be complemented with a larger number of iterations in the compression function. Because all current attacks against MD5, SHA-0, and SHA-1 become obsolete when the number of iterations is, for instance, doubled (cf. [18]), this would address the first point raised above. Thus, at the cost of making them about half as fast, all current cryptographic hash standards can be restored to their originally intended security levels with respect to collision resistance. A similar change to SHA-2 should address its perceived security problem as well. An additional problem is caused by Joux's multicollision trick, but if finding collisions is infeasible to begin with, then that is not a big deal – as long as one understands that concatenation of cryptographic hash functions does not buy much additional security.

Summarizing the above two paragraphs: we do not know precisely what we need or want, we know that by the time the result of the NIST competition is announced it is too early to fully trust the outcome if it is a new design, we can have reasonable confidence in slightly altered versions of functions we are all familiar with, and integration of the latter would be a breeze compared to incorporating a new method. The sole problems of the above naive approach – which can no doubt be strengthened considerably by a few more simple changes – seem to be that it does not address multicollisions and the slowdown by a factor of two. The cryptographic grapevine, however, finds it unreasonable to expect the same speed from cryptographic hash functions as from block ciphers and, to some extent, attributes the weaknesses of current standards to a misguided desire to make them fast (cf. [16]). With these simple-minded approaches and given the issues pointed out combined with the requirements of the marketplace, there would have been a more sustainable alternative for Suite B to sticking to vanilla SHA-2. Research on cryptographic hash functions is currently very active, but given the tight schedule of NIST's cryptographic hash function competition it becomes hard for a new standard to fully profit from new insights that are currently under development.

6 Public-key cryptography

In symmetric cryptosystems each pair of communicating parties needs to share, and occasionally refresh, a key. Bootstrapping and maintaining such a system is costly because it requires trusted channels to exchange keying material. Also, it gives rise to cumbersome key management issues, to keep track of the proper keys for each party one would be dealing with.

In principle key exchange, key management, and also encryption can all conveniently be dealt with using public-key cryptosystems. Additionally, public-key cryptosystems can be used for an application that was not envisioned in the symmetric cryptography world, namely digital signatures. A very cursory description follows. The quadratic overhead of the number of traditional symmetric keys is avoided by the fact that in a public-key cryptosystem each party has its own personal key, as opposed to sharing a key with every other party it needs to communicate with. Each of these personal keys consists of two parts: a private key and a public key, with complementary capabilities depending on the functionality the key is used for. If used for data confidentiality, the public part is used to encrypt, while the private key is required for decryption. If used for digital signatures, the private key is used to generate a signature, the validity of which can be verified using the public key. Obviously, this requires secrecy of the private key and, on the other hand, broad accessibility of the corresponding public key. For either functionality to provide k-bit security, deriving the private key from the public one should require 'effort' at least 2^k.

Complications arise due to key revocation issues, the certification mechanism that is required for public keys, and due to the fact that a single personal key should not be used for different functionalities. These are beyond the scope of this write-up.

No proven and practical approaches have been published to realize public-key cryptosystems. All practical systems proposed so far rely on certain hardness assumptions, the most popular of which are the assumptions that integer factorization and computing discrete logarithms in certain cyclic groups are hard. In the former system, RSA, the private key essentially consists of two large primes, and the public key is formed by taking their product. In the latter, a generator g of a cyclic group G of prime order and an element $h \in G$ together form the public key, while the private key is the least non-negative integer x such that $g^x = h$ (assuming the group operation in G is written multiplicatively): this x is called the discrete logarithm of h with respect to g. A large community of users may use the same infrastructure provided by the public key parts g and G, with each particular user selecting their own secret x and publishing the corresponding $h = g^x$ as their public key: the probability of selecting identical key pairs is negligible, unless the group is too small to provide adequate security. In RSA, however, no sharing of infrastructural key data is possible: each user must select their own large primes, and publish their product (generally referred to as the *RSA modulus*). In both cases keys can be efficiently generated.

It is unknown whether factoring an RSA modulus or computing $\log_g h$ is required to break either system, but doing so breaks it for sure. Thus, to assess the security of either system one needs to know how the RSA modulus size and properties of G behave as a function of the desired security level. In the case of RSA this boils down to a single question, namely how hard it is to factor integers. Based on the current state of the art in factorization algorithms one can be led to believe that a properly chosen 4096-bit modulus provides, approximately, 128-bit security. It is more contentious that higher security levels of 192 and 256 bits are provided by 8192-bit and 16384-bit moduli, respectively (cf. [10]). The popular belief that 1024-bit RSA moduli give 80 bits of security is questionable, the alleged 112-bit security of 2048-bit moduli is on the optimistic side.

For discrete logarithm based cryptosystems the situation is more complicated, because there are many possible choices for G, each with their own peculiarities with respect to the discrete logarithm problem. In all popular systems G is a prime order subgroup of the multiplicative group of a finite field or of the group of points of an elliptic curve. A first, generic requirement is that the cardinality of G must be at least 2^{2k} in order to obtain k-bit security. If G can be embedded in the multiplicative group of a finite field then the cardinality of the smallest finite field for which this is possible must satisfy very similar requirements as RSA moduli: for instance the smallest field cardinality should be a 4096-bit number to get 128-bit security. That this not only applies to the traditional finite field discrete logarithm problem but also to some elliptic curve groups, came as an unwelcome surprise to some. As a consequence the affected curves, so-called supersingular elliptic curves, were considered to be 'weak cases' (cf. [9]) and abandoned for regular public-key cryptosystems. They have since been resurrected, with different parameter settings, because of their applicability to pairing-based cryptosystems – a development that not only the authors of [9] find 'especially striking'.

Supersingular elliptic curves were not the only example were an initially trusted elliptic curve discrete logarithm problem turned out to be easier than expected. Somewhat more surprisingly, discrete logarithms in multiplicative groups of finite fields of relatively low extension degree (typically 30) also proved to be easier to solve than expected, due to their connection to certain hyperelliptic curve problems. Generally speaking, however, and avoiding the published bad cases, it is widely believed that there are no other security criteria than the two mentioned above: always a $2k$-bit prime order subgroup, and if the subgroup is embeddable in a finite field the latter's characteristic must satisfy the same requirements as RSA moduli. Given the current state of the art, it would be inappropriate to cast doubt on the strength of properly chosen elliptic curve discrete logarithm systems. It would also be wrong to give the impression that consensus has been reached. For some people there is a lingering doubt, fostered by a variety of questions and developments. To mention a few, if the subject had indeed been researched so extensively as claimed, why did it take until 2007 before a classical and practically relevant elliptic curve parameterization (now referred to as 'Edwards coordinates') resurfaced? What other 'classical' results are waiting to be rediscovered? Is it not remarkable that in certain types of elliptic curve groups the decision Diffie-Hellman problem is known to be easy, whereas there is no indication that this is the case in certain isomorphic subgroups of multiplicative groups of finite fields – despite a snide comment in [9]. Also, one keeps wondering why a group that allows so much faster than generic point counting (cf. [17]) would allow just generic discrete logarithm computation.

Suite B prescribes public-key cryptosystems just for digital signatures and key exchange. It does encryption using a hybrid scheme based on key exchange followed by symmetric encryption. Although RSA and discrete logarithm cryptosystems both support key exchange and digital signatures (and encryption), Suite B relies exclusively on elliptic curve based discrete logarithm cryptosystems. Furthermore, it specifies a small, fixed number of elliptic curve groups to be used, of 256-bit and 384-bit orders providing 128-bit and 192-bit security, respectively. The NIST version not only allows more flexibility for the group sizes but also for the elliptic curve choices. And, interestingly, NIST allows 2048-bit RSA for both public key functionalities plus, for digital signatures, 3072-bit RSA and the 2048-bit or 3072-bit versions of the US government's Digital Signature Algorithm (DSA). Furthermore, it allows the 2048-bit version of the traditional discrete logarithm based Diffie-Hellman method for key exchange.

It may be argued that the NSA put all its eggs in one basket with its limited choice of parameters for just a single type of public key cryptosystem. Most certainly, the NIST version of Suite B offers a greater degree of flexibility, both at the level of public key cryptosystem and as far as parameter choices are concerned. It is a matter of taste if a limited choice of curves is good or bad since, as also argued in [9], leaving users freedom in elliptic curve parameter choices, which may sound good, carries the risk of selecting poor curves. As far as Suite B's restriction to a single system is concerned, on the relevant website (cf. [14]) the following statement can be found:

> A key aspect of Suite B is its use of elliptic curve technology instead of classical public key technology. NSA has determined that beyond the 1024-bit public key cryptography in common use today, rather than increase key sizes beyond 1024-bits, a switch to elliptic curve technology is warranted.

This is not very informative because no reason is given why the switch is warranted. But a reason may be that for the higher than 128-bit security level required by NSA it is unclear what parameters would have to be standardized for RSA, DSA, or Diffie-Hellman. Furthermore, even if one could agree on adequate parameter choices, their size would make implementations unappealing and performance poor. Performance would also be a problem for 512-bit elliptic curve cryptosystems, which is according to [13] one of the reasons those systems are not included in Suite B, the other reason being that 192-bit security (as obtained using the 384-bit elliptic curve system) is adequate anyhow. That this results in a remarkable security level mismatch in Suite B (with 128-bit and 256-bit block cipher security levels, but 128-bit and 192-bit public key security levels) is allegedly not a concern, based on the argument that implementers prefer AES with 256-bit keys and that this large key size does not incur a serious performance penalty over the 192-bit version (cf. [13]).

It will be a long time before the last word has been said about elliptic curve cryptosystems, their security, and the particular curves selected for Suite B, unless an unexpected and for Suite B undesirable cryptanalytic elliptic curve breakthrough occurs: if that happens the public key part of Suite B would be dead. Systems that adopted the NIST version could survive[2]. Does the risk of that happening warrant the additional cost of implementing two or even three public key systems? That looks unlikely, even to some of those who, like the present author, recommended against outright adoption of elliptic curve cryptography one or two decades ago – it is a relief however, that Suite B did not jump on the pairing-based bandwagon yet.

Confidence in the security of to be standardized cryptographic technologies is obviously an important issue. But another important factor is free availability of the technologies in question. Continuing the paragraph quoted above from [14]:

[2] Unless the breakthrough has a wider effect. It could, for instance, involve the development, finally, of a truly working quantum computer. We do not venture a guess as to when that may happen.

In order to facilitate adoption of Suite B by industry, NSA has licensed the rights to 26 patents held by Certicom Inc. covering a variety of elliptic curve technology. Under the license, NSA has a right to sublicense vendors building equipment or components in support of US national security interests. Any vendor building products for national security use is eligible to receive a license from the National Security Agency. For further information on Elliptic Curve Intellectual Property Licensing please contact the Business Affairs Office of the NSA/CSS Commercial Solutions Center. For further information visit: `http://www.nsa.gov/ia/industry/cep.cfm`.

Although this begins promising, it is unclear where it leaves vendors that are not covered by the terms of the license, but that nevertheless want to build a Suite B compliant cryptographic toolkit. It is in sharp and unwelcome contrast to the other technologies standardized in Suite B, which are free of intellectual property issues. Supporting its industry in the widest sense is implied by the mission statement in the first quote in Section 1. Not doing so violates the intentions of Suite B.

7 Concluding remarks

Suite B's proposed switch to higher security levels is timely and it allows a variety of security levels that should be adequate for several decades to come. But Suite B is inflexible in the choice of cryptosystems by limiting the standard to a single system for each of the three main functionalities. The NIST variant is commendable for giving at least some choice in two of the three functionalities, while also offering greater security level diversity. Suite B's inflexibility would not be a concern if the three main components involved are beyond reproach. One of them, however, needs to be used with care when implemented in software (AES). This needs to be pointed out explicitly in the relevant standard. Another functionality is based on a design principle that is no longer fully supported by the cryptologic research community (SHA-2) and may be replaced or complemented by the result of the NIST competition. This creates confusion, hesitation, and potential incompatibilities, and may lead to costly upgrades. It could have been avoided. The third (elliptic curve cryptography) is encumbered by patents until well beyond the year 2010.

Realistically speaking the cryptographic issues are a relatively minor concern. The societal aspects surrounding information security are in much greater disarray than any of the current or future cryptographic standards. There are many problems to be solved. To address those challenges, we expect that psychologists will play a more important role than cryptologists.

References

1. J. Bos, M. Kaihara, Pollard rho on the PlayStation 3, in preparation, September 2008.
2. eSTREAM, the ECRYPT stream cipher project: `http://www.ecrypt.eu.org/stream/`.
3. The emperor's new security indicators: `http://www.usablesecurity.org/emperor/`.
4. Shay Gueron, Advanced encryption standard (AES) instructions set, white paper, Intel Corporation, April 2008.
5. P. Hofstee, personal communication, September 2008.
6. A. Joux, Multicollisions in iterated hash functions, proceedings Crypto 2004, Springer-Verlag LNCS 3152, 306–316.
7. C.S. Jutla, A.C. Pattak, A Simple and Provably Good Code for SHA Message Expansion: `http://eprint.iacr.org/2005/247`.
8. J.M. Kelsey, How to choose SHA-3, keynote at [24]
9. A.H. Koblitz, N. Koblitz, A. Menezes, Elliptic curve cryptography, the serpentine course of a paradigm shift, September 2008, `eprint.iacr.org/2008/390`.
10. A.K. Lenstra, Unbelievable security, proceedings Asiacrypt 2001, Springer-Verlag LNCS 2248, 67–86.
11. H.W. Lenstra, Rijndael for algebraists: `http://math.berkeley.edu/~hwl/papers/rijndael0.pdf`.
12. National Institute of Standards and Technology, Cryptographic hash algorithm competition: `http://www.nist.gov/hash-competition`.

13. National Institute of Standards and Technology, Suite B Cryptography: `http://csrc.nist.gov/groups/SMA/ispab/documents/minutes/2006-03/E_Barker-March2006-ISPAB.pdf`
14. National Security Agency, Fact sheet NSA Suite B Cryptography: `http://www.nsa.gov/ia/Industry/crypto_suite_b.cfm`.
15. D.A. Osvik, A. Shamir, E. Tromer, *Cache attacks and Countermeasures: the Case of AES*, August 2005, `eprint.iacr.org/2005/271`.
16. Panel discussion at [24].
17. R. Schoof, Elliptic curves over finite fields and the computation of square roots mod p, Math. Comp. **44** (1985), pp 483–494.
18. M. Stevens, personal communication, September 2008.
19. X. Wang, X. Lai, D. Feng, H. Chen, X. Yu, Cryptanalysis of the hash functions MD4 and RIPEMD, Proceedings Eurocrypt 2005, Springer-Verlag LNCS 3494, 1–18.
20. X. Wang, H. Yu, How to break MD5 and other hash functions, Proceedings Eurocrypt 2005, Springer-Verlag LNCS 3494, 19–35.
21. X. Wang, H. Yu, Y.L. Lin, Efficient collision search attacks on SHA-0, Proceedings Crypto 2005, Springer-Verlag LNCS 3621, 1–16.
22. X. Wang, Y.L. Lin, H. Yu, Finding collisions in the full SHA-1, Proceedings Crypto 2005, Springer-Verlag LNCS 3621, 17–36.
23. M. Wiener, The full cost of cryptanalytic attacks, Journal of Cryptology **17** (2004), 105–124.
24. Workshop 'Hash functions in cryptology: theory and practice', June 2–6, 2008, Leiden, The Netherlands.