International Journal of Theoretical & Applied Sciences, 2(1): 41-44(2010) ISSN : 0975-1718

On industrial needs of symmetric cryptography

Santosh Kumar Yadav, Neelima Relan and Jaspal Singh Bhatia**

Department of Mathematical Sciences, Kalindi College, University of Delhi, INDIA
**J.J.T. University, Jhunjhunu (RJ) INDIA*

(*Received 10 Jan., 2010, Accepted 27 Feb., 2010*)

ABSTRACT : Cryptography is a science as well as a technique which has its goal as applicability. The conversion of scientific results on cryptography into actual products is performed mostly by private companies and public sector organisations who wish to incorporate security features within various applications. These organisations are collectively described as "the industry" and the purpose of this paper is to provide a rough sketch of what are their current needs in the area of symmetric cryptography.

Keywords : Secure protocols, AES, SHS, Stream Cipher, Random Seeds

INTRODUCTION

Scientific research on Cryptography operates on abstract, mathematical objects such as integers or elements is a finite field. Industrial applications are more concerned about sending and receiving streams of bits or bytes. For proper interoperability, an algorithm must be specified completely and unambiguously, which means that two distinct and independent implementers should be able to produce the exact same output with the same input without resourting to "common knowledge". This paper will show the problems which the industry would like the researchers, to tackle most immediately.

Industrial needs in symmetric cryptography can be sorted into the follows categories :

(*i*) Standards;
(*ii*) Secure protocols;
(*iii*) High-performance specialised algorithms;
(*iv*) Random number generators'
(*v*) Implementation issues

STANDARDIZATION

A variety of standardization challenges are presented as follows :

A. Data representation

For symmetric cryptography, what algorithm (scientific) descriptions most often lack is a precise definition of the ordering of bits within a byte, and the ordering of bytes within a multi-byte integer value. Some standards, such as the AES [4] and SHS [3] do it correctly : they take great pain to define bit and byte ordering precisely.

Without such a precise definition, any algorithm specification is next to useless for the industry; if the algorithm is really needed (for instance, it has very good properties which no properly specified existing algorithm provides), the industrial organisation will fill in the blanks with its own proprietary definitions, and thus is interoperability forfeit.

B. Responsibilities

Apart from being clear and precise, a good standard must be *accepted* and *maintained*. For instance, the FIPS publications by the NIST [11] (such as AES and SHS) fit this description well. They are backed by the US government and as such their use is mandated throughout many government applications; conformance to these standards can be qualified as a legal obligation. Moreover, the NIST is officially responsible for the contents of the standards, and *must* publish revisions when the needs arises. Using these standards is economically justified. The RFC [8] system is not as good. RFC publication is relatively easy (and free), but most RFCs are tagged as "informational" which gives no hint on their acceptance and foreseeable future. Only those RFCs which are related to Internet protocols get a chance to become RFC "standards". RFCs are *never* modified in any way; revisions may be issued but there is no guarantee of that. A good example of this is the TLS protocol [15] and the RC4 stream cipher : RC4 is, technically, a trademark corresponding to a secret algorithm. However, there exists another algorithm (usually called "alleged RC4" or "Arcfour") which is not secret and which appears to be compatible with the official RC4. The "alleged" situation is not satisfactory, hence a new RFC describing Arcfour was scheduled, and the TLS RFC specifies that Arcfour, as described in that new document, is compatible with the official RC4. This was stated when the TLS RFC was written, back in January 1999. However the Arcfour description was never published and since nobody is responsible for this document, it cannot be said when, if ever, Arcfour will be described in a public authoritative reference.

C. Secure protocols

Cryptographic algorithms are only bricks which are combined into *protocols* which provide some high-level security features. An example of such a protocol is TLS [15], which combines symmetric encryption, MACs and asymmetric cryptography (key exchange and signature) in order to provide a confidential authenticated integrity-checked two-way tunnel for arbitrary byte streams: the underlying medium is any other two-way tunnel for byte streams. The performance and adequacy of a protocol in a specific situation is highly dependent on the protocol definition. The regular "standard" protocols are usually high-level and best suited to Internet-like communications. For

small lightweight applications (e.g. embedded mobile devices), implementers are reluctant to implement TLS or IPsec tunnelling because they cannot afford a fully functional IP implementation. For these devices special lightweight protocols must be designed which have provisions for the characteristics of the environment.

D. Encryption Modes

A raw symmetric encryption algorithm is either a block cipher or a stream cipher.

A block cipher encrypts only one block; to encrypt more, the incoming data must be split into several blocks and a special mode of encryption used. The most naive mode, the ECB mode, is known to be weak. The usual recommendation is to use CBC [6] mode, but this has the following problems :

(*i*) CBC needs an initial value which should be added to the encrypted message, thus enlarging it.

(*ii*) CBC encrypts data only if it has a length which is a multiple of the block size. If the underlying data does not have this property then some padding must be applied, which usually results in further message length increase. Some specific padding modes have been devised to avoid this problem (e.g. "ciphertext stealing", aka CTS) but their security still needs some serious analysis.

(*iii*) CBC can be interleaved by splitting the input into separate streams in order to improve throughput (*e.g.*, on Triple DES platforms that have three DES processors). But, in general, CBC cannot be made parallel. This is a problem for high-bandwidth devices.

Stream ciphers do not have padding issues, but they have security issues when a key is reused. The usual approach is to include an IV and combine it somehow with the key, but the actual combining process is rarely described and can have adverse effects if not done properly. Moreover, if the IV cannot be derived from some contextual information (*e.g.*, message sequence number), it must be sent along with the message, which increases its length. Note that a block cipher in CTR mode is only a way to make a stream cipher from a block cipher; the IV in this case is the counter initial value.

The industry needs modes of encryption which provide some or all of the following characteristics :

(*i*) little or no increase in message size;

(*ii*) precise and complete specification of proper ways to derive IVs and other values;

(*iii*) possibility of parallel implementation (for high-speed devices with specific hardware);

(*iv*) Low cost;

(*v*) possibility of encryption of very short message;

(*vi*) if possible, patent-free.

E. Combined encryption and MAC algorithm

MACs are used to provide integrity checks on data. Assurance of integrity is needed in most protocols, in order to thwart active attacks (*e.g.*, modifying data). Although such attacks are usually much more difficult to perform than simple eavesdropping, they are nonetheless increasingly important for the industry. Integrity checks can be used to provide authentication (by knowledge of a shared secret), independently of any need for confidentiality; however, it is often the case that both confidentiality and data integrity are needed. Some historical applications (*e.g.*GSM mobile phones) use encryption for data authentication by having some conventional data encrypted and checked. Such a way to build a MAC is known to be insecure in most situations; for instance, the WEP protocol, designed to protect WiFi connections, was a spectacular failure in that matter. However, industrial implementers are often reluctant to compute both encryption and MAC, because it basically doubles the cost. This is especially true for low-power mobile devices, which have limited computing and electrical power.

F. Hash Functions

A good hash function must have specific properties such as collision resistance. Although other candidates have been proposed, the most used nowadays derive from the MD family. MD4 [12] is considered as broken, and its successors MD5 [13], SHA-0 and SHA-1 are also technically broken. Some functions in the SHA family [3] (SHA-256, SHA-384 and SHA-512) are still considered secure, but the new results on SHA-0 and SHA-1 raise some concerns about the whole family.

Moreover, implementing both a block cipher and a hash function in a limited low-power device *(e.g.* a smartcard) can be troublesome, due to hard limitations on the ROM size, or die surface. It is conceivable to use a key-agile block cipher as a hash function by using the data as private key, with some chaining and padding; however, this problem has not been well studied, and has been the subject of only limited standardization (see ISO/IEC 10118-2 [9].

HIGH–PERFORMANCE ALGORITHMS

Some applications need specialised algorithms with very high constraints on performance. They can be split into roughly two groups :

A. High-speed specialised network nodes

We consider here devices which have to handle huge amounts of data; the cryptographic algorithms they use must be able to process data with a very high bandwidth and very low latency. High bandwidth is usually achieved using pipelining, and this is possible so long as the algorithm itself can be expressed as a circuit. This is true for most, if not all, block ciphers, but the mode of operation is also important because non-parallel modes such as CBC defeat pipelining. Latency cannot be reduced easily. Among block ciphers, those with little diffusion and many rounds usually imply a high latency because the critical path for each data bit must traverse many layers. Modern block ciphers are quite good in that respect (but there is always an application for which the existing method is not "good enough").

B. Low-power devices

Low-power devices are applications where cryptography must be applied by hardware which is very limited in either or both of computational power and electrical resources. Some extreme applications are RFID tags, which receive very low power through electromagnetic induction, and have very limited time to compute and send back an answer through radio waves. Other examples include Bluetooth-enabled devices which are often battery-powered and yet must sustain radio communications with medium to high data bandwidth.

Most of those applications have to ultimately rely on a general purpose 8-bit or 16-bit processor, with limited room for code and static data (in ROM) and *very* limited room for mutable data (in RAM). Most modern cryptographic algorithms are designed as a compromise between high speed on workstations and "generically acceptable performance" for low-power embedded devices. It so happens that some applications require better than "generically acceptable"; the industry needs some algorithms which can be relied upon for security and which perform well on limited platforms.

RANDOM NUMBER GENERATORS

Random number generation is difficult, especially for cryptographic purposes, because the "quality" of the produced random data cannot be measured. Usual statistical tests provide only a very superficial view of the problem; cryptographic protocols require computational unpredictability. A random number generator which does not pas successfully the statistical tests is bad indeed; but a generator which does, cannot be thus declared "good".

A cryptographic random number generator is usually the combination of some *seed*, which is a random value provided externally, and a pseudo-random number generator (PRNG) which expands that seed into an arbitrarily long stream of bits which are computationally indistinguishable from random bits. The two main problems are:

(*i*) how to make a good seed;
(*ii*) how to define and implement a good PRNG.

A. Random Seeds

A seed is expressed as a value in a format which is suitable for the purpose of the PRNG which will be used; it usually is a stream of bits of some specified length (160 bits is common practice). The seed must have an *entropy* which is good enough to thwart exhaustive search; actually, entropy can be defined as the average cost for an exhaustive search to succeed.

Entropy estimation and concentrations are the two main parts of the problem of producing seeds. Entropy estimation is about measuring how much "unpredictability" can be attached to the result of measuring a physical event. For instance, a coin flip yields one bit of entropy– if we assume that the coin is not biased in any way. Entropy sources commonly used vary, depending on the context :

(*i*) In embedded devices such as smartcards, a specialized hardware random number generator is mandatory; usual generators use a reverse-biased PN junction, implemented with Zener diodes or a smart combination of transistors.
(*ii*) In workstations the physical events are usually precise timings of external interruptions, wich correspond to network activity, key strokes, etc. Some processors include a hardware random number generator which uses a technology equivalent to those used in smarcards.
(*iii*) There are some environment where almost no source of randomness is supplied, for instance virtual machines used to run applets. Those machines are *meant* to not have any randomness. There exists a seed gathering procedure which has been published for the Java Virtual Machine; the idea is to measure the efficiency of the thread schedular, because that efficiency should depend on the actual load of the host multitasking machine.

To sum up, the industry has the following needs with regards to seed generation for cryptographic purposes :

(*i*) recipes for gathering random data in various situations;
(*ii*) accurate estimators for the entropy thus obtained;
(*iii*) secure ways to concentrate random data, using either a hash function, a block cipher or a stream cipher.

B. Pseudo-Random Number Generator (PRNG)

The main difference is that a PRNG has no concept of input bandwidth, just output bandwidth. An encryption system which just encrypts an endless stream of zeroes is supposed to be a good PRNG; similarly, a PRNG can be transformed into a reasonable stream cipher by using the seed as a key, and combining the output bits with the data using a bitwise exclusive-or. Parallelism is still important : it can be exploited by hardware implementations to provide better output bandwidth; but the same effect can be achieved by including several instances of the generator, working on several seeds which have been derived from a master seed using another PRNG.

Another detail which makes PRNG performance easier than encryption performance is that a PRNG works only for producing random bytes, whereas an encryption system definition must be wary of the feasibility and performance of the corresponding decryption system.

What the industry needs here is a list of research-approved ways to build a PRNG using a primitive cryptographic operation, where that operation may be either a hash function, a block cipher or a stream cipher.

IMPLEMENTATION ISSUES

A. Side-Channel Attacks

The implementation of cryptographic algorithms is a complex matter, because it deals with security and this is not easily measured. An implementation which passes all

test vectors may still have security issues related to, for instance, side-channel attacks. Such attacks exploit some data leakage due to an implementation detail; the usual leak mediums are timing (algorithm computation time is not independent of the processed data and secret key) and power consumption (especially for smartcards or other devices which have an external power supply). Defending against such leaks is not easy, especially with power consumption for smartcards, because power is provided externally, by a potentially hostile entity. Devising generic ways to implement primitive operations in ways which do not leak private data are an active research area, and much work still needs to be done.

B. Testing : The implementation of symmetric algorithm can be very difficult to test

However, some algorithms use Operations which can be incorrectly implemented in subtle ways.

One such example is the DFC block cipher [7]. That algorithm includes a modular affine transform; the reduction is performed module $2^{64} + 13$. It is relatively difficult to implement that reduction in a way which is both correct for every input and also efficient. It is actually easy to implement it in a way which is correct for most inputs but incorrect with a probability of 2^{-64} for random input data. Random test vectors have a very low probability of catching such an error; specific test vectors must be devised, so as to exercise, the specific input values which may be handled improperly.

Many algorithms use hard-coded look-up tables, where the tables have been chosen for specific properties. A typing error could corrupt one table entry and remain undetected by the test vectors if none of them uses that table with the corresponding entry.

In brief, for an algorithm specification to be properly usable by the industry, appropriate validation procedures (test vectors, mostly) should be included. These procedures must be defined so as to be very likely to catch the most common implementation errors.

CONCLUSION AND FUTURE TRENDS

We can conclude the most common industrial needs in symmetric cryptography as follows :

(*i*) precise standards actively maintained and supported by large, and if possible public sector organisations;

(*ii*) new enhanced modes of operations, both for encryption and combined modes which provide both encryption and authentication;

(*iii*) precise guidelines on the handling of related data such as IVs;

(*iv*) specialised algorithms for use in contexts were usual algorithms do not provide adequate performance, especially low-power embedded devices;

(*v*) analysed hardware random number generators with accurate entropy estimators;

(*vi*) high-performance secure PRNGs based on various algorithms such as block ciphers, hash functions and stream ciphers.

(*vii*) guidelines for methods which reduce secret data leakage through side channels;

(*viii*) proper validation procedures for all standard algorithms;

(*ix*) migration strategies.

Particularly there is a strong need for high-level constructions (hashing, PRNG, MAC, etc) which use a block cipher as the unique underlying cryptographic operation. This is for small, lightweight applications which cannot afford the concurrent implementation of a block cipher and a hash function.

REFERENCES

[1] M. Bellare, P. Rogaway, and D. Wagner. The EAX mode of operation. In *Fast Software Encryption – FSE* (2004), Volume 3017 of *L.N.C.S.,* pages 389-407, Springer-Verlag, (2004).

[2] D. Bleichenbacher, Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS#1. In *Advances in Cryptology – CRYPTO '98,* volume 1462 of *L N C S,* page 1-12, Springer-Verlag, (1998).

[3] FIPS 180-2, Secure hash Standard, Federal Information Processing Standards Publication 180-2, U.S. Department of Commerce / N.I.S.T(2002).

[4] EIPS 197, Advanced Encryption Standard, Federal Information Processing Standards Publication 197, U.S. Department of Commerce / N.I.S.T(2001).

[5] FIPS 46-3, Data Encryption Standard, Federal Information Processing standards Publication 46-3, U.S. Department of Commerce / National Bureau of Standards(1999).

[6] FIPS 81, DES Modes of Operation, Federal Information Processing Standards Publication 81, U.S. Department of Commerce / National Bureau of Standards(1980).

[7] H. Gilbert, M. Girault, P. Hoogvorst, F. Noilhan, T. Pornin, G. Poupard, J. Stern, and S. Vaudenay, Decorrelated Fast Cipher : an AES candidate, (1998).

[8] The Internet Society (ISOC), RFC-Editor.

[9] ISO / IEC 10118-2, Information technology-Security techniques - Hash-functions-Part 2 : Hash-functions, using an n-bit block cipher algorithm. International Organization for Standardization, (2000).

[10] T. Kohno, J. Viega, and D. Whiting, CWC : A High-Performance Conventional authenticated Encryption Mode. In *Fast Software Encryption – FSE* (2004), Volume 3017 of *L N C S,* pages 408-426, Springer-Verlag, (2004).

[11] National Institute of Standards and Technology (NIST), Federal Information Processing Standards (FIPS).

[12] RFC 1320, The MD4 Message Digest Algorithm, Internet Request for Comments 1320, R.L. Rivest(1992).

[13] RFC 1321, The MD5 Message Digest Algorithm, Internet Request for Comments 1321, R.L. Rivest(1992).

[14] RFC 2104, HMAC; Keyed-Hashing for Message Authentication, Internet Request for Comments 2104, H. Krawczyk and M. Bellare and R. Canetti(1997).

[15] RFC 2246, The TLS Protocol, Internet Request for Comments 2246, T. Dierks and C. Allen(1999).

[16] RFC 3610, Counter with CBC-MAC (CCM), Internet Request for Comments 3610, D. Whiting, R. Housley and N. Ferguson(2003).

[17] P. Rogaway, M. Bellare, and J. Black, OCB; A Block-Cipher Mode of Operation for Efficient Authenticated Encryption, *ACM Trans, Information System and Security,* **6**(3)**:** 365-403(2003).