XMX: A Firmware-oriented Block Cipher Based on Modular Multiplications

[Published in E. Biham, Ed., Fast Software Encrytion, vol. 1267 of Lecture Notes in Computer Science, pp. 166–171, Springer-Verlag, 1997.]

David M'Raïhi¹, David Naccache¹, Jacques Stern², and Serge Vaudenay²

¹ Gemplus - Cryptography Department 1 place de la Méditerranée, 95206, Sarcelles CEDEX, France {100145.2261, 100142.3240}@compuserve.com ² École Normale Supérieure 45 rue d'Ulm, 75230, Paris CEDEX 5, France {jacques.stern, serge.vaudenay}@gemplus.com

Abstract. This paper presents xmx, a new symmetric block cipher optimized for public-key libraries and microcontrollers with arithmetic coprocessors. xmx has no S-boxes and uses only modular multiplications and xors. The complete scheme can be described by a couple of compact formulae that offer several interesting time-space trade-offs (number of rounds/key-size for constant security).

In practice, xmx appears to be tiny and fast: 136 code bytes and a 121 kilo-bits/second throughput on a Siemens SLE44CR80s smart-card (5 MHz oscillator).

1 Introduction

Since efficiency and flexibility are probably the most appreciated design criteria, block ciphers were traditionally optimized for either software (typically SAFER [4]) or hardware (DES [2]) implementation. More recently, autonomous agents and object-oriented technologies motivated the design of particularly tiny codes (such as TEA [9], 189 bytes on a 68HC05) and algorithms adapted to particular programming languages such as PERL.

Surprisingly, although an ever-increasing number of applications gain access to arithmetic co-processors [5] and public-key libraries such as BSAFE, MIR-ACL, BIGNUM [8] or ZEN [1], no block cipher was specifically designed to take advantage of such facilities.

This paper presents $\times m \times (\text{xor-multiply-xor})$, a new symmetric cipher which uses public-key-like operations as confusion and diffusion means. The scheme does not require S-boxes or permutation tables, there is virtually no key-schedule and the code itself (when relying on a co-processor or a library) is extremely compact and easy to describe.

xmx is firmware-suitable and, as such, was specifically designed to take a (carefully balanced) advantage of hardware and software resources.

2 David M'Raïhi, David Naccache, Jacques Stern, and Serge Vaudenay

2 The Algorithm

2.1 Basic operations

xmx is an iterated cipher, where a keyed primitive f is applied r times to an ℓ -bit cleartext m and a key k to produce a ciphertext c.

Definition 1. Let $f_{a,b}(m) = (m \circ a) \cdot b \mod n$ where:

$$x \circ y = \begin{cases} x \oplus y & \text{if } x \oplus y < n \\ x & \text{otherwise} \end{cases}$$

and n is an odd modulus.

Property: $a \circ b$ is equivalent to $a \oplus b$ in most cases (when $n \leq 2^{\ell}$, and $\{a, b\}$ is uniformly distributed, $\Pr[a \circ b = a \oplus b] = n/2^{\ell}$).

Property: For all a and b, $a \circ b \circ b = a$.

f can therefore be used as a simply invertible building-block (a < n implies $a \circ b < n)$ in iterated ciphers :

Definition 2. Let n be an ℓ -bit odd modulus, $m \in \mathbb{Z}_n$ and k be the key-array $k = \{a_1, b_1, \ldots, a_r, b_r, a_{r+1}\}$ where $a_i, b_i \in \mathbb{Z}_n^*$ and $gcd(b_i, n) = 1$.

The block-cipher xmx is defined by:

$$\mathsf{xmx}(k,m) = (f_{a_r,b_r}(f_{a_{r-1},b_{r-1}}(\dots(f_{a_1,b_1}(m))\dots))) \circ (a_{r+1})$$

and:

$$\mathsf{xmx}^{-1}(k,c) = (f_{a_1,b_1}^{-1}(f_{a_2,b_2}^{-1}(\dots(f_{a_r,b_r}^{-1}(c \circ a_{r+1}))\dots)))$$

2.2 Symmetry

A crucially practical feature of xmx is the symmetry of encryption and decryption. Using this property, xmx and xmx^{-1} can be computed by the same procedure:

Lemma 1.

$$k^{-1} = \{a_{r+1}, b_r^{-1} \bmod n, a_r, \dots, b_1^{-1} \bmod n, a_1\} \Rightarrow \mathsf{xmx}^{-1}(k, x) = \mathsf{xmx}(k^{-1}, x)$$

Since the storage of k requires $(2r + 1)\ell$ bits, xmx schedules the encryption and decryption arrays k and k^{-1} from a single ℓ -bit key s:

$$k(s) = \{s, s, \dots, s, s, s \oplus s^{-1}, s, s^{-1}, \dots, s, s^{-1}\}$$

where $k^{-1}(s) = k(s^{-1})$.

For a couple of security reasons (explicited *infra*) s must be generated by the following procedure (where w(s) denotes the Hamming weight of s):

XMX: A Firmware-oriented Block Cipher Based on Modular Multiplications

```
1. Pick a random s \in \mathbb{Z}_n^{\star} such that \frac{\ell}{2} - \log_2 \ell < w(s) < \frac{\ell}{2} + \log_2 \ell
2. If \gcd(s, n) \neq 1 or \ell - \log_2 s \ge 2 go to 1.
```

3. output the key-array $k(\bar{s)} = \{s, s, \dots, s, s, s \oplus s^{-1}, s, s^{-1}, \dots, s, s^{-1}\}$

Although equally important, the choice of n is much less restrictive and can be conducted along three engineering criteria: prime moduli will greatly simplify key generation $(\gcd(b_i, n) = 1 \text{ for all } i)$, RSA moduli used by existing applications may appear attractive for memory management reasons and dense moduli will increase the probability $\Pr[a \circ b = a \oplus b]$.

As a general guideline, we recommend to keep n secret in all real-life applications but assume its knowledge for the sake of academic research.

Security 3

xmx's security was evaluated by targeting a weaker scheme (wxmx) where $\circ \cong \oplus$ and $k = (s, s, s, \dots, s, s, \dots, s, s, s)$.

Using the trick $u \oplus v = u + v - 2(u \wedge v)$ for eliminating xors and defining:

 $h_i(x) = ((\dots (x \oplus a_1) \cdot b_1 \mod n \dots) \oplus a_{i-1}) \cdot b_{i-1} \mod n$

we get by induction:

 $\mathsf{wxmx}(k, x) = b'_1 \cdot x + a_1 \cdot b'_1 \dots + a_{r+1} - 2(g_1(x) \cdot b'_1 + \dots + g_{r+1}(x)) \mod n$

where $b'_i = b_i \cdots b_r \mod n$ and $g_i(x) = h_i(x) \wedge a_i$.

Consequently,

wxmx
$$(k, x) = b'_1 \cdot x + b - 2g(x) \mod n$$
 where $b = a_1 \cdot b'_1 + a_2 \cdot b'_2 \dots + a_{r+1}$

and
$$g(x) = g_1(x) \cdot b'_1 + g_2(x) \cdot b'_2 + \ldots + g_{r+1}(x) \mod n$$
.

3.1The number of rounds

When r = 1, the previous formulae become $g_2(x) = h_2(x) \wedge s$ and

 $\mathsf{wxmx}(k, x) = ((x \oplus s) \cdot s \mod n) \oplus s = x s + s^2 + s - 2(g_1(x)s + g_2(x)) \mod n$

Assuming that $w(\delta)$ is low, we have (with a significantly high probability):

$$g_1(x+\delta) = (x+\delta) \wedge s = g_1(x) \mod n$$
.

Therefore, selecting δ such that $s \wedge \delta = 0 \Rightarrow g_1(x \oplus \delta) = g_1(x)$, we get $\mathsf{wxmx}(k, x \oplus \delta) - \mathsf{wxmx}(k, x) = (x \oplus \delta - x) \cdot s - 2 \left(s \wedge h_2(x \oplus \delta) - s \wedge h_2(x) \right) \mod n .$ Plugging $\delta = 2$ and an x such that $x \wedge \delta = 0$ into this equation, we get:

 $\mathsf{wxmx}(k, x \oplus \delta) - \mathsf{wxmx}(k, x) = 2\left(s - s \wedge h_2(x+2) + s \wedge h_2(x)\right) \mod n \ .$

Since $h_2(x) = s \cdot x + s^2 - 2g_1(x) \mod n$ (where $g_1(x) = x \wedge s$), it follows that $h_2(x)$ and $h_2(x+2)$ differ only by a few bits. Consequently, information about s leaks out and, in particular, long sequences of zeros or ones (with possibly the first and last bits altered) can be inferred from the difference wxmx $(k, x \oplus \delta) - wxmx(k, x)$.

In the more general setting (r > 1), we have

$$\mathsf{wxmx}(k, x \oplus \delta) - \mathsf{wxmx}(k, x) = (x \oplus \delta - x)s^r + 2e(x, \delta, s) \mod n$$

where $e(x, \delta, s)$ is a linear form with coefficients of the form $\alpha \wedge s - \beta \wedge s$.

Defining $\Delta = \{ \mathsf{wxmx}(k, x \oplus \delta) - \mathsf{wxmx}(k, x) \}$, we get $\|\Delta\| < 2^{rw(s)}$ since Δ is completely characterized by s.

The difference will therefore leak again whenever:

$$2^{rw(s)} < 2^{\ell} \quad \Rightarrow \quad r < \frac{\ell}{w(s)} \quad . \tag{1}$$

3.2 Key-generation

4

The weight of s: Since g(x) is a polynomial which coefficients (b'_i) are all bitwise smaller than s, the variety of g(x) is small when w(s) is small. In particular, when $w(s) < \frac{80}{r+1}$, less than 2^{80} such polynomials exist.

A 2⁴⁰-pair known plaintext attack would therefore extract s^r from:

$$\operatorname{wxmx}(k, y) - \operatorname{wxmx}(k, x) = (y - x) \cdot s^r \mod n$$

using the birthday paradox (the same g(x) should have been used twice). One can even obtain collisions on g with higher probability by simply choosing pairs of similar plaintexts. Using [7] (refined in [6]), these attacks require almost no memory.

Since a similar attack holds for \overline{s} when w(s) is big $(x \oplus y = x + 2(\overline{x} \wedge y) - y)$, w(s) must be rather close to $\ell/2$ and (1) implies that r must at least equal three to avoid the attack described in section 3.1.

The size of s: Chosen plaintext attacks on wxmx are also possible when s is too short: if s m < n after r iterations, s can be recovered by encrypting $m = 0_{\ell}$ since wxmx $(k, 0_{\ell}) = b - 2g(x)$ and g's coefficients are all bounded by s.

Observing that $0 \leq \mathsf{wxmx}(k, 0_\ell) - s^{r+1} \leq s \cdot 2^r$, we have:

$$0 \leq s - \sqrt[r+1]{\operatorname{wxmx}(k, 0_\ell)} < \frac{1}{r+1} \quad \Rightarrow \quad s = \left\lceil \sqrt[r+1]{\operatorname{wxmx}(k, 0_\ell)} \right\rceil \ .$$

More generally, encrypting short messages with short keys may also reveal s. As an example, let $\ell = 256$, r = 4, $s = 0_{176}|s'$ and $m = 0_{176}|m'$ where s' and m'are both 80-bit long. Since $\Pr[x \oplus s = x + s] = (3/4)^{80} \cong 2^{-33}$ when s is 80-bit long, a gcd between ciphertexts will recover s faster than exhaustive search. XMX: A Firmware-oriented Block Cipher Based on Modular Multiplications

3.3 Register size

Since the complexity of section 3.1's attack must be at least 2^{80} , we have:

$$\sqrt{2^{r \cdot w(s)}} > 2^{80}$$

and considering that $w(s) \cong \ell/2$, the product $r\ell$ must be at least 320.

r = 4 typically requires $\ell > 80$ (brute force resistance implies $\ell > 80$ anyway) but an inherent $2^{\ell/2}$ -complexity attack is still possible since wxmx is a (keyed) permutation over ℓ -bit numbers, which average cycle length is $2^{\ell/2}$ (given an iteration to the order $2^{\ell/2}$ of wxmx(k, x), one can find x with significant probability).

 $\ell = 160$ is enough to thwart these attacks.

4 Implementation

Standard implementations should use xmx with r = 8, $\ell = 512$, $n = 2^{512} - 1$ and

$$k = \{s, s, s, s, s, s, s, s, s, s \oplus s^{-1}, s, s^{-1}, s, s^{-1}, s, s^{-1}, s, s^{-1}\}$$

while high and very-high security applications should use $\{r = 12, \ell = 768, n = 2^{786} - 1\}$ and $\{r = 16, \ell = 1024, n = 2^{1024} - 1\}$.

A recent prototype on a Siemens SLE44CR80s results in a tiny (136 bytes) and performant code (121 kilo-bits/second throughput with a 5 MHz oscillator) and uses only a couple of 64-byte buffers.

The algorithm is patent-pending and readers interested in test-patterns or a copy of the patent application should contact the authors.

5 Further Research

As most block-ciphers xmx can be adapted, modified or improved in a variety of ways: the round output can be subjected to a constant permutation such as a circular rotation or the chunk permutation $\pi(ABCD) \rightarrow BADC$ where each chunk is 128-bit long (since $\pi(\pi(x)) = x$, xmx's symmetry will still be preserved). Other variants replace modular multiplications by point additions on an elliptic curve (ecxmx) or implement protections against [3] (taxmx).

It is also possible to define f on two ℓ -bit registers L and R such that:

$$f(L_1, R_1) = \{L_2, R_2\}$$

where

$$L_2 = R_1$$
 and $R_2 = L_1 \oplus ((R_1 \oplus k_2) \cdot k_1 \mod n).$

and the inverse function is:

 $R_1 = L_2, L_1 = R_2 \oplus ((R_1 \oplus k_2) \cdot k_1 \mod n) = R_2 \oplus ((L_2 \oplus k_2) \cdot k_1 \mod n)$

Since such designs modify only one register per round we recommend to increase r to at least twelve and keep generating s with xmx's original key-generation procedure.

6 David M'Raïhi, David Naccache, Jacques Stern, and Serge Vaudenay

6 Challenge

It is a tradition in the cryptographic community to offer cash rewards for successful cryptanalysis. More than a simple motivation means, such rewards also express the designers' confidence in their own schemes. As an incentive to the analysis of the new scheme, we therefore offer (as a souvenir from FSE'97...) 256 Israeli *Shkalim* and 80 *Agorot* (*n* is the smallest 256-bit prime starting with 80 ones) to the first person who will degrade *s*'s entropy by at least 56 bits in the instance:

$$r = 8, \ell = 256$$
 and $n = (2^{80} - 1) \cdot 2^{176} + 157$

but the authors are ready to carefully evaluate and learn from any feedback they get.

References

- F. Chabaud and R. Lercier, *The ZEN library*, http://lix.polytechnique.fr/ ~zen/
- 2. FIPS PUB 46, 1977, Data Encryption Standard.
- P. Kocher, Timing attacks in implementations of Diffie-Hellman, RSA, DSS and other systems, Advances in Cryptology - CRYPTO '96, LNCS 1109, 1996, pp. 104-113.
- 4. J. Massey, SAFER K-64: a byte oriented block cipher algorithm, Fast Software Encryption, Cambridge Security Workshop, 1993, LNCS 809, pp. 1-17.
- D. Naccache and D. M'Raïhi, Cryptographic smart cards, IEEE Micro, June 1996, vol. 16, no. 3, pp. 14-23.
- P. van Oorschot and M. J. Wiener, Parallel collision search with application to hash functions and discrete logarithms, 2nd ACM Conference on Computer and Communication Security, Fairfax, Virginia, ACM Press, 1994, pp. 210-218.
- J-J. Quisquater and J-P. Delescaille, How easy is collision search? Application to DES, Advances in Cryptology - EUROCRYPT'89, LNCS 434, 1990, pp. 429-434.
- B. Serpette, J. Vuillemenin and J. C. Hervé, *BIGNUM: a portable and efficient package for arbitrary-precision arithmetic*, PRL Research Report #2, 1989, ftp://ftp.digital.com/pub/DEC/PRL/research-reports/PRL-RR-2.ps.Z.
- D. J. Wheeler and R. M. Needham, TEA, a tiny encryption algorithm, Fast Software Encryption, Leuven, LNCS 1008, 1994, pp. 363-366.