# 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.]<br>David M'Raïhi ${ }^{1}$, David Naccache ${ }^{1}$, Jacques Stern ${ }^{2}$, and Serge Vaudenay ${ }^{2}$<br>${ }^{1}$ Gemplus - Cryptography Department 1 place de la Méditerranée, 95206, Sarcelles Cedex, France<br>\{100145.2261, 100142.3240\}@compuserve.com<br>${ }^{2}$ École Normale Supérieure<br>45 rue d'Ulm, 75230, Paris cedex 5, France<br>\{jacques.stern, serge.vaudenay\}@gemplus.com


#### Abstract

This paper presents $\times m \times$, 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 68 HC 05 ) 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, MIRACL, BIGNUM [8] or ZEN [1], no block cipher was specifically designed to take advantage of such facilities.

This paper presents $\times m \times$ (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 The Algorithm

### 2.1 Basic operations

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

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

$$
x \circ y=\left\{\begin{array}{cl}
x \oplus y & \text { if } x \oplus y<n \\
x & \text { otherwise }
\end{array}\right.
$$

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, $\left.\operatorname{Pr}[a \circ b=a \oplus b]=n / 2^{\ell}\right)$.
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 $\ell$-bit odd modulus, $m \in \mathbb{Z}_{n}$ and $k$ be the key-array $k=\left\{a_{1}, b_{1}, \ldots, a_{r}, b_{r}, a_{r+1}\right\}$ where $a_{i}, b_{i} \in \mathbb{Z}_{n}^{*}$ and $\operatorname{gcd}\left(b_{i}, n\right)=1$.

The block-cipher xmx is defined by:

$$
\mathrm{xm} \times(k, m)=\left(f_{a_{r}, b_{r}}\left(f_{a_{r-1}, b_{r-1}}\left(\ldots\left(f_{a_{1}, b_{1}}(m)\right) \ldots\right)\right)\right) \circ\left(a_{r+1}\right)
$$

and:

$$
\operatorname{xmx}^{-1}(k, c)=\left(f_{a_{1}, b_{1}}^{-1}\left(f_{a_{2}, b_{2}}^{-1}\left(\ldots\left(f_{a_{r}, b_{r}}^{-1}\left(c \circ a_{r+1}\right)\right) \ldots\right)\right)\right)
$$

### 2.2 Symmetry

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

## Lemma 1.

$k^{-1}=\left\{a_{r+1}, b_{r}^{-1} \bmod n, a_{r}, \ldots, b_{1}^{-1} \bmod n, a_{1}\right\} \Rightarrow \mathrm{xmx}^{-1}(k, x)=\mathrm{xm} \times\left(k^{-1}, x\right)$.
Since the storage of $k$ requires $(2 r+1) \ell$ bits, $\mathrm{xm} \times$ schedules the encryption and decryption arrays $k$ and $k^{-1}$ from a single $\ell$-bit key $s$ :

$$
k(s)=\left\{s, s, \ldots, s, s, s \oplus s^{-1}, s, s^{-1}, \ldots, s, s^{-1}\right\}
$$

where $k^{-1}(s)=k\left(s^{-1}\right)$.
For a couple of security reasons (explicited infra) $s$ must be generated by the following procedure (where $w(s)$ denotes the Hamming weight of $s$ ):

1. Pick a random $s \in Z_{n}^{\star}$ such that $\frac{\ell}{2}-\log _{2} \ell<w(s)<\frac{\ell}{2}+\log _{2} \ell$
2. If $\operatorname{gcd}(s, n) \neq 1$ or $\ell-\log _{2} s \geq 2$ go to 1 .
3. output the key-array $k(s)=\left\{s, s, \ldots, s, s, s \oplus s^{-1}, s, s^{-1}, \ldots, s, s^{-1}\right\}$

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 $\left(\operatorname{gcd}\left(b_{i}, n\right)=1\right.$ for all $\left.i\right)$, RSA moduli used by existing applications may appear attractive for memory management reasons and dense moduli will increase the probability $\operatorname{Pr}[a \circ b=a \oplus b]$.

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.

## 3 Security

$x m x$ 's security was evaluated by targeting a weaker scheme ( $w x m x$ ) where $\circ \cong \oplus$ and $k=(s, s, s, \ldots, s, s, \ldots, s, s, s)$.

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

$$
h_{i}(x)=\left(\left(\ldots\left(x \oplus a_{1}\right) \cdot b_{1} \bmod n \ldots\right) \oplus a_{i-1}\right) \cdot b_{i-1} \bmod n
$$

we get by induction:

$$
\mathrm{wxm} \times(k, x)=b_{1}^{\prime} \cdot x+a_{1} \cdot b_{1}^{\prime} \ldots+a_{r+1}-2\left(g_{1}(x) \cdot b_{1}^{\prime}+\ldots+g_{r+1}(x)\right) \bmod n
$$

where $b_{i}^{\prime}=b_{i} \cdots b_{r} \bmod n$ and $g_{i}(x)=h_{i}(x) \wedge a_{i}$.
Consequently,

$$
\begin{gathered}
\mathrm{wxm} \times(k, x)=b_{1}^{\prime} \cdot x+b-2 g(x) \bmod n \text { where } b=a_{1} \cdot b_{1}^{\prime}+a_{2} \cdot b_{2}^{\prime} \ldots+a_{r+1} \\
\text { and } g(x)=g_{1}(x) \cdot b_{1}^{\prime}+g_{2}(x) \cdot b_{2}^{\prime}+\ldots+g_{r+1}(x) \bmod n
\end{gathered}
$$

### 3.1 The number of rounds

When $r=1$, the previous formulae become $g_{2}(x)=h_{2}(x) \wedge s$ and
$\mathrm{wxmx}(k, x)=((x \oplus s) \cdot s \bmod n) \oplus s=x s+s^{2}+s-2\left(g_{1}(x) s+g_{2}(x)\right) \bmod n$
Assuming that $w(\delta)$ is low, we have (with a significantly high probability):

$$
g_{1}(x+\delta)=(x+\delta) \wedge s=g_{1}(x) \bmod n
$$

Therefore, selecting $\delta$ such that $s \wedge \delta=0 \quad \Rightarrow \quad g_{1}(x \oplus \delta)=g_{1}(x)$, we get $\mathrm{wxmx}(k, x \oplus \delta)-\mathrm{wxmx}(k, x)=(x \oplus \delta-x) \cdot s-2\left(s \wedge h_{2}(x \oplus \delta)-s \wedge h_{2}(x)\right) \bmod n$.

Plugging $\delta=2$ and an $x$ such that $x \wedge \delta=0$ into this equation, we get:

$$
\mathrm{wxmx}(k, x \oplus \delta)-\mathrm{wxmx}(k, x)=2\left(s-s \wedge h_{2}(x+2)+s \wedge h_{2}(x)\right) \bmod n .
$$

Since $h_{2}(x)=s \cdot x+s^{2}-2 g_{1}(x) \bmod n\left(\right.$ where $\left.g_{1}(x)=x \wedge s\right)$, 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 $\mathrm{wxmx}(k, x \oplus \delta)-$ $\mathrm{wxmx}(k, x)$.
In the more general setting $(r>1)$, we have

$$
\mathrm{w} \times \mathrm{m} \times(k, x \oplus \delta)-\mathrm{w} \times \mathrm{m} \times(k, x)=(x \oplus \delta-x) s^{r}+2 e(x, \delta, s) \bmod n
$$

where $e(x, \delta, s)$ is a linear form with coefficients of the form $\alpha \wedge s-\beta \wedge s$.
Defining $\Delta=\{\mathrm{wxmx}(k, x \oplus \delta)-\mathrm{w} \times \mathrm{mx}(k, x)\}$, we get $\|\Delta\|<2^{r w(s)}$ since $\Delta$ is completely characterized by $s$.

The difference will therefore leak again whenever:

$$
\begin{equation*}
2^{r w(s)}<2^{\ell} \Rightarrow r<\frac{\ell}{w(s)} . \tag{1}
\end{equation*}
$$

### 3.2 Key-generation

The weight of $s$ : Since $g(x)$ is a polynomial which coefficients $\left(b_{i}^{\prime}\right)$ 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 $2^{40}$-pair known plaintext attack would therefore extract $s^{r}$ from:

$$
\mathrm{w} \times \mathrm{m} \times(k, y)-\mathrm{w} \times \mathrm{m} \times(k, x)=(y-x) \cdot s^{r} \bmod n
$$

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 $\bar{s}$ when $w(s)$ is big $(x \oplus y=x+2(\bar{x} \wedge y)-y)$, $w(s)$ 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 $\operatorname{wxmx}\left(k, 0_{\ell}\right)=b-2 g(x)$ and $g$ 's coefficients are all bounded by $s$.

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

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

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

### 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 $w \times m \times$ is a (keyed) permutation over $\ell$-bit numbers, which average cycle length is $2^{\ell / 2}$ (given an iteration to the order $2^{\ell / 2}$ of $\mathrm{w} \times \mathrm{mx}(k, x)$, one can find $x$ with significant probability).

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

## 4 Implementation

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

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

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

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 $x m \times$ 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 $\pi(\mathrm{ABCD}) \rightarrow$ BADC where each chunk is 128 -bit long (since $\pi(\pi(x))=x, \mathrm{xmx}$ '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 $\ell$-bit registers $L$ and $R$ such that:

$$
f\left(L_{1}, R_{1}\right)=\left\{L_{2}, R_{2}\right\}
$$

where

$$
L_{2}=R_{1} \quad \text { and } \quad R_{2}=L_{1} \oplus\left(\left(R_{1} \oplus k_{2}\right) \cdot k_{1} \bmod n\right) .
$$

and the inverse function is:

$$
R_{1}=L_{2}, L_{1}=R_{2} \oplus\left(\left(R_{1} \oplus k_{2}\right) \cdot k_{1} \bmod n\right)=R_{2} \oplus\left(\left(L_{2} \oplus k_{2}\right) \cdot k_{1} \bmod n\right)
$$

Since such designs modify only one register per round we recommend to increase $r$ to at least twelve and keep generating $s$ with $\times m \times$ 's original keygeneration procedure.

## 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:

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

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

## References

1. F. Chabaud and R. Lercier, The ZEN library, http://lix.polytechnique.fr/ ~zen/
2. FIPS PUB 46, 1977, Data Encryption Standard.
3. 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.
5. D. Naccache and D. M'Raïhi, Cryptographic smart cards, IEEE Micro, June 1996, vol. 16, no. 3, pp. 14-23.
6. P. van Oorschot and M. J. Wiener, Parallel collision search with application to hash functions and discrete logarithms, $\mathbf{2}^{\text {nd }}$ ACM Conference on Computer and Communication Security, Fairfax, Virginia, ACM Press, 1994, pp. 210-218.
7. 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.
8. B. Serpette, J. Vuillemenin and J. C. Hervé, BIGNUM: a portable and efficient package for arbitrary-precision arithmetic, PRL Research Report $\sharp \mathbf{2}, 1989$, ftp://ftp.digital.com/pub/DEC/PRL/research-reports/PRL-RR-2.ps.Z.
9. D. J. Wheeler and R. M. Needham, TEA, a tiny encryption algorithm, Fast Software Encryption, Leuven, LNCS 1008, 1994, pp. 363-366.
