Introduction to symmetric crypto

D. J. Bernstein

How HTTPS protects connection:

- Public-key encryption system encrypts *one* secret message: a random 256-bit session key.
- Public-key signature system stops NSAITM attacks.
- Fast **authenticated cipher** uses the 256-bit session key to protect further messages.

Some cipher history

1973, and again in 1974:
U.S. National Bureau of
Standards solicits proposals
for a Data Encryption Standard.

Some cipher history

1973, and again in 1974:
U.S. National Bureau of
Standards solicits proposals
for a Data Encryption Standard.

1975: NBS publishes IBM DES
proposal. 64-bit block, 56-bit key.

Some cipher history

1973, and again in 1974: U.S. National Bureau of Standards solicits proposals for a Data Encryption Standard.

1975: NBS publishes IBM DES proposal. 64-bit block, 56-bit key.

1976: NSA meets Diffie and Hellman to discuss criticism. Claims "somewhere over $400,000,000" to break a DES key; "I don't think you can tell any Congressman what's going to be secure 25 years from now."

1977: DES is standardized.

1977: Diffie and Hellman publish detailed design of \$20,000,000 machine to break hundreds of DES keys per year.

1977: DES is standardized.

1977: Diffie and Hellman publish detailed design of $20,000,000 machine to break hundreds of DES keys per year.

1978: Congressional investigation into NSA influence concludes "NSA convinced IBM that a reduced key size was sufficient".

1977: DES is standardized.

1977: Diffie and Hellman publish detailed design of $20,000,000 machine to break hundreds of DES keys per year.

1978: Congressional investigation into NSA influence concludes "NSA convinced IBM that a reduced key size was sufficient".

1983, 1988, 1993: Government reaffirms DES standard.

1977: DES is standardized.

1977: Diffie and Hellman publish detailed design of $20,000,000 machine to break hundreds of DES keys per year.

1978: Congressional investigation into NSA influence concludes "NSA convinced IBM that a reduced key size was sufficient".

1983, 1988, 1993: Government reaffirms DES standard.

Researchers publish new cipher proposals and security analysis.

1997: U.S. National Institute of Standards and Technology (NIST, formerly NBS) calls for proposals for Advanced Encryption Standard. 128-bit block, 128/192/256-bit key.

1997: U.S. National Institute of Standards and Technology (NIST, formerly NBS) calls for proposals for Advanced Encryption Standard. 128-bit block, 128/192/256-bit key.

1998: 15 AES proposals.

1997: U.S. National Institute of Standards and Technology (NIST, formerly NBS) calls for proposals for Advanced Encryption Standard. 128-bit block, 128/192/256-bit key.

1998: 15 AES proposals.

1998: EFF builds "Deep Crack" for under \$250000 to break hundreds of DES keys per year.

1997: U.S. National Institute of Standards and Technology (NIST, formerly NBS) calls for proposals for Advanced Encryption Standard. 128-bit block, 128/192/256-bit key.

1998: 15 AES proposals.

1998: EFF builds "Deep Crack" for under $250000 to break hundreds of DES keys per year.

1999: NIST selects five AES finalists: MARS, RC6, Rijndael, Serpent, Twofish.

2000: NIST, advised by NSA, selects Rijndael as AES.

"Security was the most important factor in the evaluation"—Really?

2000: NIST, advised by NSA, selects Rijndael as AES.

"Security was the most important factor in the evaluation"—Really?

"Rijndael appears to offer an *adequate* security margin. . . . Serpent appears to offer a *high* security margin."

2000: NIST, advised by NSA, selects Rijndael as AES.

"Security was the most important factor in the evaluation"—Really?

"Rijndael appears to offer an *adequate* security margin. . . . Serpent appears to offer a *high* security margin."

2004–2008: eSTREAM competition for stream ciphers.

2000: NIST, advised by NSA, selects Rijndael as AES.

"Security was the most important factor in the evaluation"—Really?

"Rijndael appears to offer an *adequate* security margin. . . . Serpent appears to offer a *high* security margin."

2004–2008: eSTREAM competition for stream ciphers.
2007–2012: SHA-3 competition.

2000: NIST, advised by NSA, selects Rijndael as AES.

"Security was the most important factor in the evaluation"—Really?

"Rijndael appears to offer an *adequate* security margin. . . . Serpent appears to offer a *high* security margin."

2004–2008: eSTREAM competition for stream ciphers.
2007–2012: SHA-3 competition.
2013–2019: CAESAR competition.

2000: NIST, advised by NSA, selects Rijndael as AES.

"Security was the most important factor in the evaluation"—Really?

"Rijndael appears to offer an *adequate* security margin. . . . Serpent appears to offer a *high* security margin."

2004–2008: eSTREAM competition for stream ciphers.
2007–2012: SHA-3 competition.
2013–2019: CAESAR competition.
2019–now: NISTLWC competition.

Main operations in AES:
add round key to block;
apply **substitution box**
$x \mapsto x^{254}$ in $\mathbf{F}_{256}$
to each byte in block;
linearly mix bits across block.

Main operations in AES:
add round key to block;
apply **substitution box**
$x \mapsto x^{254}$ in $\mathbf{F}_{256}$
to each byte in block;
linearly mix bits across block.

Extensive security analysis.
Even in a post-quantum world,
no serious threats to AES-256
in a strong security model,
"multi-target SPRP security".

Main operations in AES:
add round key to block;
apply **substitution box**
$x \mapsto x^{254}$ in $\mathbf{F}_{256}$
to each byte in block;
linearly mix bits across block.

Extensive security analysis.
Even in a post-quantum world,
no serious threats to AES-256
in a strong security model,
"multi-target SPRP security".

So why isn't AES-256 the end
of the symmetric-crypto story?

Google Security Blog

The latest news and insights from Google on security and safety on the Internet

Speeding up and strengthening HTTPS connections for Chrome on Android

April 24, 2014

Posted by Elie Bursztein, Anti-Abuse Research Lead

Earlier this year, we deployed a new TLS cipher suite in Chrome that operates three times faster than AES-GCM on devices that don't have AES hardware acceleration, including most Android phones, wearable devices such as Google Glass and older computers. This improves user experience, reducing latency and saving battery life by cutting down the amount of time spent encrypting and decrypting data.

To make this happen, Adam Langley, Wan-Teh Chang, Ben Laurie and I began implementing new algorithms -- ChaCha 20 for symmetric encryption and Poly1305 for authentication -- in OpenSSL and NSS in March 2013. It was a complex effort that required implementing a new abstraction layer in OpenSSL in order to support the Authenticated Encryption with Associated Data (AEAD) encryption mode properly. AEAD enables encryption and authentication to happen concurrently, making it easier to use and optimize than older, commonly-used modes such as CBC. Moreover, recent attacks against RC4 and CBC also prompted us to make this change.

The benefits of this new cipher suite include:

Date: 2018-08-06 22:32:51
Message-ID: 20180806223300.11389
[Download message RAW]

From: Eric Biggers <ebiggers@google.co

Hi all,

(Please note that this patchset is a tr
it to be merged quite yet!)

It was officially decided to *not* allo
encryption [1]. We've been working to
storage encryption to entry-level Andro
"Android Go" devices sold in developing
these devices still ship with no encryp
have to use older CPUs like ARM Cortex-
Cryptography Extensions, making AES-XTS

As we explained in detail earlier, e.g
challenging problem due to the lack of
the very strict performance requiremen
suitable for practical use in dm-crypt
Speck, in this day and age the choice o
has a large political element, restrict

Therefore, we (well, Paul Crowley did t
encryption mode, HPolyC. In essence, H
ChaCha stream cipher for disk encryptio
paper here: https://eprint.iacr.org/20

1-1-ebiggers () kernel ! org 10

m>

rue RFC, i.e. we're not ready for

ow Android devices to use Speck
find an alternative way to bring
oid devices like the inexpensive
g countries. Unfortunately, often
ption, since for cost reasons they
-A7; and these CPUs lack the ARMv8
S much too slow.

. in [2], this is a very
encryption algorithms that meet
ts, while still being secure and
and fscrypt. And as we saw with
of cryptographic primitives also
ting the options even further.

the real work) designed a new
HPolyC makes it secure to use the
on. HPolyC is specified by our
18/720.pdf ("HPolyC:

Google Security Blog

The latest news and insights from Google on security and safety on the Internet

Introducing Adiantum: Encryption for the Next Billion Users

February 7, 2019

Posted by Paul Crowley and Eric Biggers, Android Security & Privacy Team

Storage encryption protects your data if your phone filesystem design.

Where AES is used, the conventional solution for disk encryption is to use the XTS or CBC-ESSIV modes of operation, which are length-preserving. Currently Android supports AES-128-CBC-ESSIV for full-disk encryption and AES-256-XTS for file-based encryption. However, when AES performance is insufficient there is no widely accepted alternative that has sufficient performance on lower-end ARM processors.

To solve this problem, we have designed a new encryption mode called Adiantum. Adiantum allows us to use the ChaCha stream cipher in a length-preserving mode, by adapting ideas from AES-based proposals for length-preserving encryption such as HCTR and HCH. On ARM Cortex-A7, Adiantum encryption and decryption on 4096-byte sectors is about 10.6 cycles per byte, around 5x faster than AES-256-XTS.

AES performance seems limited
in both hardware and software
by small 128-bit block size,
heavy S-box design strategy.

AES performance seems limited in both hardware and software by small 128-bit block size, heavy S-box design strategy.

AES software ecosystem is complicated and dangerous. Fast software implementations of AES S-box often leak secrets through timing.

AES performance seems limited in both hardware and software by small 128-bit block size, heavy S-box design strategy.

AES software ecosystem is complicated and dangerous. Fast software implementations of AES S-box often leak secrets through timing.

Picture is worse for high-security authenticated ciphers. 128-bit block size limits "PRF" security. Workarounds are hard to audit.

ChaCha creates safe systems
with much less work than AES.

ChaCha creates safe systems with much less work than AES.

More examples of how symmetric primitives have been improving speed, simplicity, security:

PRESENT is better than DES.

Skinny is better than Simon and Speck.

Keccak, BLAKE2, Ascon are better than MD5, SHA-0, SHA-1, SHA-256, SHA-512.

Authentication details

Standardize a prime $p = 1000003$.

Assume sender knows independent uniform random secrets
$r_1 \in \{0, 1, \ldots, 999999\}$,
$r_2 \in \{0, 1, \ldots, 999999\}$,
$\vdots$
$r_5 \in \{0, 1, \ldots, 999999\}$,
$s_1 \in \{0, 1, \ldots, 999999\}$,
$\vdots$
$s_{100} \in \{0, 1, \ldots, 999999\}$.

Assume receiver knows the same secrets $r_1, r_2, \ldots, r_5, s_1, \ldots, s_{100}$.

Assume receiver knows the same secrets $r_1, r_2, \ldots, r_5, s_1, \ldots, s_{100}$.

Later: Sender wants to send 100 messages $m_1, \ldots, m_{100}$, each m_n having 5 components $m_{n,1}, m_{n,2}, m_{n,3}, m_{n,4}, m_{n,5}$ with $m_{n,i} \in \{0, 1, \ldots, 999999\}$.

Assume receiver knows the same secrets $r_1, r_2, \ldots, r_5, s_1, \ldots, s_{100}$.

Later: Sender wants to send 100 messages $m_1, \ldots, m_{100}$, each m_n having 5 components $m_{n,1}, m_{n,2}, m_{n,3}, m_{n,4}, m_{n,5}$ with $m_{n,i} \in \{0, 1, \ldots, 999999\}$.

Sender transmits 30-digit $m_{n,1}, m_{n,2}, m_{n,3}, m_{n,4}, m_{n,5}$ together with an **authenticator**

$$(m_{n,1} r_1 + \cdots + m_{n,5} r_5 \bmod p) + s_n \bmod 1000000$$

and the message number n.

e.g. $r_1 = 314159$, $r_2 = 265358$,
$r_3 = 979323$, $r_4 = 846264$,
$r_5 = 338327$, $s_{10} = 950288$,
$m_{10} = 000006\ 000007\ 000000\ 000000\ 000000$:

e.g. $r_1 = 314159$, $r_2 = 265358$,
$r_3 = 979323$, $r_4 = 846264$,
$r_5 = 338327$, $s_{10} = 950288$,
$m_{10} =$ 000006 000007 000000 000000 000000:

Sender computes authenticator
$(6r_1 + 7r_2 \bmod p)$
$\quad + s_{10} \bmod 1000000 =$
$(6 \cdot 314159 + 7 \cdot 265358$
$\quad \bmod 1000003)$
$\quad + 950288 \bmod 1000000 =$
$742451 + 950288 \bmod 1000000 =$
692739.

e.g. $r_1 = 314159$, $r_2 = 265358$,
$r_3 = 979323$, $r_4 = 846264$,
$r_5 = 338327$, $s_{10} = 950288$,
$m_{10} =$ 000006 000007 000000 000000 000000:

Sender computes authenticator

$(6r_1 + 7r_2 \bmod p)$
$\quad + s_{10} \bmod 1000000 =$
$(6 \cdot 314159 + 7 \cdot 265358$
$\quad \bmod 1000003)$
$\quad + 950288 \bmod 1000000 =$
$742451 + 950288 \bmod 1000000 =$
692739.

Sender transmits
10 000006 000007 000000 000000 000000 692739.

A MAC using fewer secrets

Instead of choosing independent

$r_1, r_2, \ldots, r_5, s_1, \ldots, s_{100}$,

choose $r, s_1, s_2, \ldots, s_{100}$.

A MAC using fewer secrets

Instead of choosing independent $r_1, r_2, \ldots, r_5, s_1, \ldots, s_{100}$, choose $r, s_1, s_2, \ldots, s_{100}$.

Sender transmits 30-digit $m_{n,1}, m_{n,2}, m_{n,3}, m_{n,4}, m_{n,5}$ together with an authenticator

$$(m_{n,1}r + \cdots + m_{n,5}r^5 \bmod p) + s_n \bmod 1000000$$

and the message number n.

i.e.: take $r_i = r^i$ in previous

$$(m_{n,1}r_1 + \cdots + m_{n,5}r_5 \bmod p) + s_n \bmod 1000000.$$

e.g. $r = 314159$, $s_{10} = 265358$,

$m_{10} = 000006\,000007\,000000\,000000\,000000$:

e.g. $r = 314159$, $s_{10} = 265358$,
$m_{10} =$ 000006 000007 000000 000000 000000:

Sender computes authenticator

$(6r + 7r^2 \bmod p)$
$\quad + s_{10} \bmod 1000000 =$
$(6 \cdot 314159 + 7 \cdot 314159^2$
$\quad \bmod 1000003)$
$\quad + 265358 \bmod 1000000 =$
$953311 + 265358 \bmod 1000000 =$
218669.

e.g. $r = 314159$, $s_{10} = 265358$,
$m_{10} =$ 000006 000007 000000 000000 000000:

Sender computes authenticator
$(6r + 7r^2 \bmod p)$
$\quad + s_{10} \bmod 1000000 =$
$(6 \cdot 314159 + 7 \cdot 314159^2$
$\quad \bmod 1000003)$
$\quad + 265358 \bmod 1000000 =$
$953311 + 265358 \bmod 1000000 =$
218669.

Sender transmits
authenticated message
10 000006 000007 000000 000000 000000 218669.

Security analysis

Attacker's goal:
Find n', m', a' such that
$m' \neq m_{n'}$ but $a' =$
$(m'(r) \bmod p) + s_{n'} \bmod 1000000$.
Here $m'(x) = \sum_i m'[i]x^i$.

Security analysis

Attacker's goal:
Find n', m', a' such that
$m' \neq m_{n'}$ but $a' =$
$(m'(r) \bmod p) + s_{n'} \bmod 1000000$.
Here $m'(x) = \sum_i m'[i]x^i$.

Obvious attack:
Choose any $m' \neq m_1$.
Choose uniform random a'.
Success chance $1/1000000$.

Security analysis

Attacker's goal:
Find n', m', a' such that
$m' \neq m_{n'}$ but $a' =$
$(m'(r) \bmod p) + s_{n'} \bmod 1000000$.
Here $m'(x) = \sum_i m'[i]x^i$.

Obvious attack:
Choose any $m' \neq m_1$.
Choose uniform random a'.
Success chance $1/1000000$.

Can repeat attack.
Each forgery has chance
$1/1000000$ of being accepted.

More subtle attack:

Choose $m' \neq m_1$ so that the polynomial $m'(x) - m_1(x)$ has 5 distinct roots $x \in \{0, 1, \ldots, 999999\}$ modulo p. Choose $a' = a$.

More subtle attack:

Choose $m' \neq m_1$ so that the polynomial $m'(x) - m_1(x)$ has 5 distinct roots $x \in \{0, 1, \ldots, 999999\}$ modulo p. Choose $a' = a$.

e.g. $m_1 = (100, 0, 0, 0, 0)$, $m' = (125, 1, 0, 0, 1)$: $m'(x) - m_1(x) = x^5 + x^2 + 25x$ which has five roots mod p: $0, 299012, 334447, 631403, 735144$.

More subtle attack:

Choose $m' \neq m_1$ so that the polynomial $m'(x) - m_1(x)$ has 5 distinct roots $x \in \{0, 1, \ldots, 999999\}$ modulo p. Choose $a' = a$.

e.g. $m_1 = (100, 0, 0, 0, 0)$, $m' = (125, 1, 0, 0, 1)$: $m'(x) - m_1(x) = x^5 + x^2 + 25x$ which has five roots mod p: $0, 299012, 334447, 631403, 735144$.

Success chance $5/1000000$.

Actually, success chance can be above 5/1000000.

Actually, success chance can be above $5/1000000$.

Example: If $m_1(334885) \bmod p \in \{1000000, 1000001, 1000002\}$ then a forgery $(1, m', a_1)$ with $m'(x) = m_1(x) + x^5 + x^2 + 25x$ also succeeds for $r = 334885$; success chance $6/1000000$.
Reason: 334885 is a root of $m'(x) - m_1(x) + 1000000$.

Actually, success chance can be above $5/1000000$.

Example: If $m_1(334885) \bmod p$ $\in \{1000000, 1000001, 1000002\}$ then a forgery $(1, m', a_1)$ with $m'(x) = m_1(x) + x^5 + x^2 + 25x$ also succeeds for $r = 334885$; success chance $6/1000000$.
Reason: 334885 is a root of $m'(x) - m_1(x) + 1000000$.

Can have as many as 15 roots of $(m'(x) - m_1(x)) \cdot$ $(m'(x) - m_1(x) + 1000000) \cdot$ $(m'(x) - m_1(x) - 1000000)$.

Do better by varying a'?

Do better by varying a'?

No. Easy to prove: Every choice of (n', m', a') with $m' \neq m_{n'}$ has chance $\leq 15/1000000$ of being accepted by receiver.

Do better by varying a'?

No. Easy to prove: Every choice of (n', m', a') with $m' \neq m_{n'}$ has chance $\leq 15/1000000$ of being accepted by receiver.

Underlying fact: ≤ 15 roots of $(m'(x) - m_1(x) - a' + a_1) \cdot$
$(m'(x) - m_1(x) - a' + a_1 + 10^6) \cdot$
$(m'(x) - m_1(x) - a' + a_1 - 10^6)$.

Do better by varying a'?

No. Easy to prove: Every choice of (n', m', a') with $m' \neq m_{n'}$ has chance $\leq 15/1000000$ of being accepted by receiver.

Underlying fact: ≤ 15 roots of $(m'(x) - m_1(x) - a' + a_1) \cdot (m'(x) - m_1(x) - a' + a_1 + 10^6) \cdot (m'(x) - m_1(x) - a' + a_1 - 10^6)$.

Warning: very easy to break the oversimplified authenticator $(m_n[1] + \cdots + m_n[5]r^4 \bmod p)$
$\quad + s_n \bmod 1000000$:
solve $m'(x) - m_1(x) = a' - a_1$.

Scaled up for serious security:

Poly1305 uses 128-bit r's,
with 22 bits cleared for speed.
Adds $s_n \bmod 2^{128}$.

Scaled up for serious security:

Poly1305 uses 128-bit r's, with 22 bits cleared for speed. Adds s_n mod 2^{128}.

Assuming $\leq L$-byte messages: Each forgery succeeds for $\leq 8 \lceil L/16 \rceil$ choices of r. Probability $\leq 8 \lceil L/16 \rceil / 2^{106}$.

Scaled up for serious security:

Poly1305 uses 128-bit r's,
with 22 bits cleared for speed.
Adds s_n mod 2^{128}.

Assuming $\leq L$-byte messages:
Each forgery succeeds for
$\leq 8\lceil L/16\rceil$ choices of r.
Probability $\leq 8\lceil L/16\rceil/2^{106}$.

D forgeries are all rejected
with probability
$\geq 1-8D\lceil L/16\rceil/2^{106}$.

Scaled up for serious security:

Poly1305 uses 128-bit r's,
with 22 bits cleared for speed.
Adds s_n mod 2^{128}.

Assuming $\leq L$-byte messages:
Each forgery succeeds for
$\leq 8\lceil L/16 \rceil$ choices of r.
Probability $\leq 8\lceil L/16 \rceil / 2^{106}$.

D forgeries are all rejected
with probability
$\geq 1 - 8D\lceil L/16 \rceil / 2^{106}$.

e.g. 2^{64} forgeries, $L = 1536$:
Pr[all rejected] ≥ 0.9999999998.

Authenticator is still secure
for variable-length messages,
if different messages are
different polynomials mod p.

Authenticator is still secure for variable-length messages, if different messages are different polynomials mod p.

Split string into 16-byte chunks, maybe with smaller final chunk; append 1 to each chunk; view as little-endian integers in $\{1, 2, 3, \ldots, 2^{129}\}$.
Multiply first chunk by r, add next chunk, multiply by r, etc., last chunk, multiply by r, mod $2^{130} - 5$, add s_n mod 2^{128}.