Security and Cryptography 2022

Mirosław Kutyłowski

XII. CRYPTO &COMMUNICATION SECURITY

part 2 -- from the second lecture

GCM (The Galois/Counter Mode)

background:

- popular as replacement for CBC mode (due to attacks presented!) and weaknesses of RC4 (now forbidden in TLS)
- fundamental critics already before standardization
- finally (April 2018) Google decided to remove it until April 2019
- operations over GF(2^{128}), addition in the field represented by $\oplus$

Computation:

1. $H\!:=\!\mathrm{Enc}_K(0^{128})$

2. $Y_0\!:=\!\mathrm{IV}||0^{31}1$ if length of IV should be 96

   or $Y_0\!:=\!\mathrm{GHASH}(H,\{\},\mathrm{IV})$

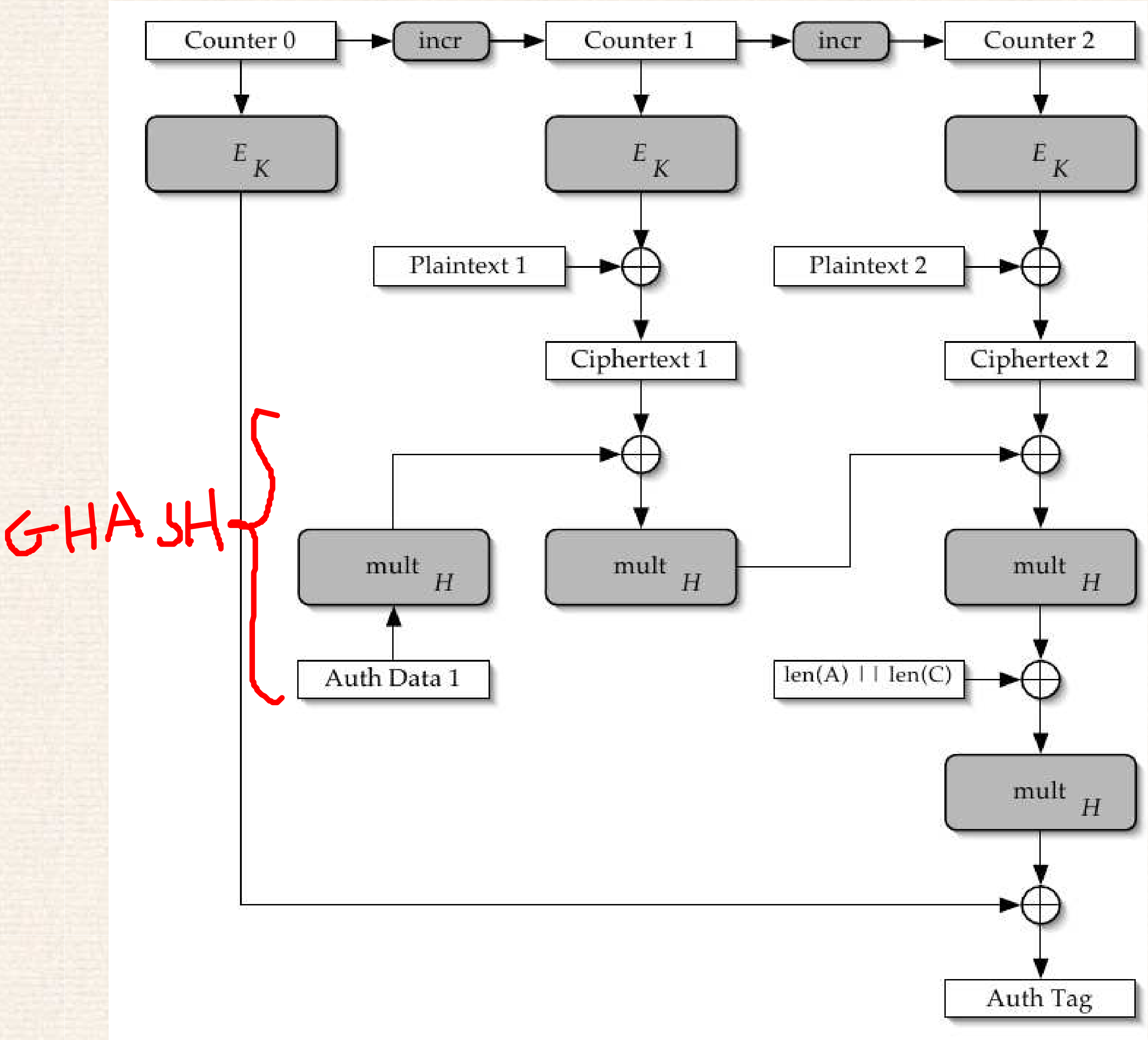
3. $Y_i\!:=\!\mathrm{incr}(Y_{i-1})$ for $i=1,\ldots,n$ (counter computation)

4. $C_i := P_i \oplus \mathrm{Enc}_K(Y_i)$ for $i=1,\ldots,n-1$ (counter based encryption)

5. $C_n^* := P_n \oplus \mathrm{MSB}_u(\mathrm{Enc}_K(Y_n))$ (the last block need not to be full)

6. $T := \mathrm{MSB}_t(\mathrm{GHASH}(H,A,C)) \oplus \mathrm{Enc}_K(Y_0)$

associated data


Counter 0
incr
Counter 1
incr
Counter 2
E_K
E_K
E_K
Plaintext 1
Plaintext 2
Ciphertext 1
Ciphertext 2
GHASH
mult_H
mult_H
mult_H
Auth Data 1
len(A) || len(C)
mult_H
Auth Tag

Details of computation of the tag

$\mathrm{GHASH}(H, A, C) = X_{m+n+1}$ where m is the length of authenticating information A, and:

X_i equals:

0	for $i = 0$
$(X_{i-1} \oplus A_i) \cdot H$	for $i = 1, ..., m-1$
$((X_{i-1} \oplus (A_m^* \| \| 0^{128-v})) \cdot H$	for $i = m$
$(X_{i-1} \oplus C_i) \cdot H$	for $i = m+1, ..., m+n-1$
$((X_{m+n-1} \oplus (C_m^* \| \| 0^{128-u})) \cdot H$	for $i = m+n$
$((X_{m+n} \oplus (\mathrm{len}(A) \| \mathrm{len}(C))) \cdot H$	for $i = m+n$+1

Decryption:

1. $H := \mathrm{Enc}_K(0^{128})$

2. $Y_0 := \mathrm{IV}||0^{31}1$ if length of IV should be 96

   or $Y_0 := \mathrm{GHASH}(H, \{\}, \mathrm{IV})$

3. $T' := \mathrm{MSB}_t(\mathrm{GHASH}(H, A, C)) \oplus \mathrm{Enc}_K(Y_0)$, is $T = T'$?

4. $Y_i := \mathrm{incr}(Y_{i-1})$ for $i = 1, \ldots, n$

5. $P_i := C_i \oplus \mathrm{Enc}_K(Y_i)$ for $i = 1, \ldots, n$

6. $P_n^* := C_n^* \oplus \mathrm{MSB}_u(\mathrm{Enc}_K(Y_n))$

Fundamental flaws (by Nils Ferguson)

– engineering disadvantages: message size up to $2^{36} - 64$ bytes, arbitrary bit length (instead of byte length)

– collisions of IV: the same pseudorandom string for encryptions

– collisions of Y_0 also possible. Due to birthday paradox 2^{64} executions might be enough for 128-bit values, for massive use in TLS the number of executions 2^{64} is maybe a threat

Ferguson attack via linear behavior

– authenticating tag computed as leading bits of $T = K_0 + \sum_{i=1}^{N} F_i \cdot H^i$ where each F_i is known but H is secret

– representing elements of $\mathrm{GF}(2^{128})$: X – as an abstract element of the field, $\mathrm{Poly}(X)$ – as a polynomial over $\mathrm{GF}(2)$ with coefficients $X_0,\ X_1,\ \ldots.,\ X_{127}$, multiplication in the field=multiplication of polynomials modulo a polynomial of degree 128

– multiplication by a constant D : $X \to D \cdot X$ can be expressed by multiplication by a matrix:

$$(D \cdot X)^T = M_D \cdot X^T \qquad \text{where } M_D \quad \text{has size } 128 \times 128$$

– squaring is linear: $(A+B)^2 = A^2 + B^2$ (field of characteristic 2), so (see the next page)

$$(X^2)^T = M_S \cdot X^T$$

where M_S is a fixed 128×128 matrix **(important point for the weakness!)**

element Z of the field treated as a formal polynomial:

$$a_0 + a_1 x + \dots + a_{127} x^{127}$$

then

$$(a_0 + \dots + a_{127} x^{127})^2 = a_0 + a_1 \cdot x^{2 \cdot 1} + \dots + a_{127} \cdot x^{2 \cdot 127}$$

note that $a_i^2 = a_i$ since $a_i \in \{0, 1\}$.

the polynomial on the right side has to be reduced modulo a polynomial of degree 128 (this is how we define operations in this field)

everything are linear operations -- they can be translated into a multiplication by a matrix

– the goal is to find a collision, i.e. C' such that

$$\sum_{i=1}^{N} C_i \cdot H^i = \sum_{i=1}^{N} C_i' \cdot H^i$$

or its leading bits (taken to MAC) are the same. Then authentication would fail – one could change the bits in a ciphertext C

– let $C_i - C_i' = E_i$, so we look for a nonzero solution to $\sum_{i=1}^{N} E_i \cdot H^i = 0$

– we confine ourselves to $E_i = 0$ except for i which is a power of 2. Let $D_i = E_{2^i}$. Let $2^n = N$

– we have to to find a solution for

$$E^T = \sum_{i=1}^{n} M_{D_i} \cdot (M_S)^i \cdot H^T$$

where E is an error vector that should become 0

– let

$$A_D = \sum_{i=1}^{n} M_{D_i} \cdot (M_S)^i$$

– then we have $E^T = A_D \cdot H^T$

– write equations to force a row of A_D to be a row of zeros (then in the result the bit of E corresponding to this row is 0), there is an equation for each bit, so 128 linear equations for the whole row

– there are $128 \cdot n$ free variables describing the values D_i (128 for each D_i)

– find a nonzero solution describing the values of D_i so that $n-1$ rows of A_D become rows of zeroes

– consider messages of length 2^{17}, $D_0 = 0$ due to issues like not changing the length

– $D_1, D_2, .., D_{17}$ can be chosen so that 16 rows of A_D are zero,

– GCM used with 32 bits MAC, so still 16 bits might be non-zero, so the chance of forgery is 2^{-16}

$$\text{16 row}\left\{\begin{bmatrix} 0 \\ \hline A_D \\ \\ \end{bmatrix}\right. \cdot \begin{bmatrix} H^T \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \hline \\ \\ \end{bmatrix} \begin{array}{l} \}\,16 \\ \\ \end{array} \Big\}\ \text{32 bits MAC}$$

One step further: Recovering $\boldsymbol{H}$

- from a collision we have 16 linear equations for H, so we may describe H by a sequence of 112 unknown bits H' and expression

$$H^T = X \cdot H'^T$$

where X is a matrix 128x112.

$$E^T = A_D \cdot X \cdot H'^T$$

- now repeat the same with H' - the attack is easier as there are only 112 bits and not 128, there are 112 equations per row and $17 \cdot 128$ free variables, so one can zeroize 19 rows and get the chance of forgery of 2^{-13}. If succeeds, then 13 new variables of H known.
- repeat until all bits of H known.
- **finally, if $\boldsymbol{H}$ is known it is possible to forge MAC for any random ciphertext $\boldsymbol{C}$** – a disaster!

ChCha

- stream cipher, Chacha extends a 256 bit stream key into 2^{64} randomly accessible streams, each of 2^{64} blocks of 64 bytes
- Daniel Berstein's construction,
- used by Google also RFC, in libraries (OpenSSL,...)
- variant of SALSA from European ECRYPT competition
- faster than AES
- working on four 32-bit words

— quarter-round of SALSA 20 for inputs a, b, c, d

1. $b := b \text{ xor } (a + d) \lll 7$
2. $c := c \text{ xor } (b + a) \lll 9$
3. $d := a \text{ xor } (c + b) \lll 13$
4. $a := a \text{ xor } (d + c) \lll 18$

— quarter-round of ChaCha20 (better diffusion)

1. $a := a + b$; $d := d \,\text{xor}\, a$; $d := d \lll 16$
2. $c := c + d$; $b := b \,\text{xor}\, c$; $b := b \lll 12$
3. $a := a + b$; $d := d \,\text{xor}\, a$; $d := d \lll 8$
4. $c := c + d$; $b := b \,\text{xor}\, c$; $b := b \lll 7$

— Chacha matrix 4x4: (where 'input' = 'block counter'+nonce)

const	const	const	const
key	key	key	key
key	key	key	key
input	input	input	input

— round: 8 quarter-rounds:

— 4 quarter rounds on: 1st column, 2nd column, 3rd column, 4th column

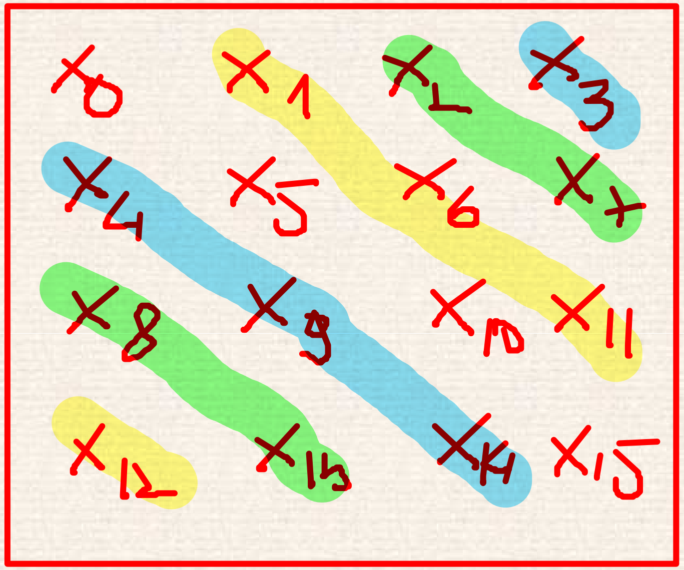
— quarter-round on diagonals

$\mathrm{QUARTERROUND}(x0, x5, x10, x15),$

$\mathrm{QUARTERROUND}(x1, x6, x11, x12)$

$\mathrm{QUARTERROUND}(x2, x7, x8, x13)$

$\mathrm{QUARTERROUND}(x3, x4, x9, x14)$



— ChaCha20 - 20 rounds

Poly1035

- designed by Bernstein, no patent, fa
- MAC algorithm, 16 byte MAC
- variable message length, 16 byte AES key, 16 byte additional key r, 16 byte nonce
- works with AES, not weaker than AES, but if AESails, then use a different encryption scheme
- the only way to break Poly is to brak AES
- per message overhead is low
- no long lookup tables, it fits into cache memory even if multiple keys used
- keys: k - for AES, r - little endian 128-bit number
- some limitations on r because of efficiency of implementation

  $r = r_0 + r_1 + r_2 + r_3$ where

$r_0 \in \{0, 1, ..., 2^{28} - 1\}$, $r_1/2^{32} \in \{0, 4, 8, 12, ..., 2^{28} - 4\}$,...

- nonces: 16 bit, encrypted by AES
- message: divided into 16 byte chunks. Each chunk treated as a 17-byte number with little-endian, where the most significant byte is an added 1 or 0, the result for a message is: $c_1, ..., c_q$
- authenticator

$$(((c_1 r^q + c_2 r^{q-1} + \ldots + c_q r^1) \bmod 2^{130} - 5) + \mathrm{AES}_k\,(\mathrm{nonce})) \bmod 2^{128}$$

denoted also $H_r(m) + \mathrm{AES}_k(\mathrm{nonce})$

- $2^{130} - 5$ is a prime,
- a nonce must be usely once
- security: for random messages m, m' of length L pbb that $H_r(m) = H_r(m') + g$ is at most $8\lceil L/16 \rceil / 2^{108}$ (all differentials have small probability)

TLS 1.3 (August 2018)

- list of symmetric algorithm contain now only AEAD (authenticated Encryption with Associated Data)
- separating key agreement and authentication from record protection
- static DH and RSA for negotiation of keys removed
- after ServerHello all handshake messages are encrypted
- key derivation function HKDF (hash key derivation function: first derive PRK via hashing of the shared secret+salt+user input, from PRK derive the secrets by hashing with sequence number)
- handshake state machine restructured to be more consistent, no superfluous messages
- elliptic curve algorithms in base specification, EdDSA included, point format negotiation

removed (one point format)

- RSA padding changed to RSASSA-PSS
- no support for some elliptic curves, MD5, SHA-224
- no compression
- prohibiting RC4 and SSL negotiation for backwards compatibility
- negotiation mechanism removed, instead a version list provided in an extension
- authentication with DH, or PSK (pre-shared key), or DH with PSK
- session resumption with PSK
- added: Chacha20 stream cipher with Poly1305 authentication code
- Ed25519 and Ed448 digital signature algorithms added

x25519 and x448 key exchange protocols added

“Certified Lies”

– rogue certificates + MitM attack: the user believes that he is directed elsewhere

– no control over root CA's worldwide, indicated either by operating system or the browser

– compelled assistance from CA's ?

ROGUE Certificates and MD5

- target: create a certificate (webserver, client) that has not been issued by CA
- not forging a signature contained in the certificate but:

  i. find two messages that $\mathrm{Hash}(M_0) = \mathrm{Hash}(M_1)$ and M_0 as well as M_1 have some common prefix that you expect in a certificate (e.g. the CA name)

  ii. submit a request corresponding to M_0, get a certificate with the signature over $\mathrm{Hash}(M_0)$

  iii. copy the signature from the certificate concerning M_0 to a certificate based on M_1

- problems: some data in M_0 are to be guessed: sequential number, validity period,

  some other are known in advance: distinguished name, ...

RSA key

request		used
legitimate website certificate		rogue CA certificate
serial number issuing CA validity period	=	serial number issuing CA validity period
domain name	chosen prefixes	rogue CA name 1024 bit RSA public key extensions “CA=true”
2048 RSA public key	collision bits	tumor
extension “CA=false”	identical suffix	

Table.

- finding M_0 and M_1 has to be fast (otherwise the guess about the serial number and validity will fail) - e.g. a day over the weekend

finding M_0, M_1 with collision requires some time even if MD5 broken

- attack on MD5, general picture:

message A		message B
prefix P		prefix P'
padding S_r		padding S_r'
birthday blocks S_b		birthday blocks S_b'
near-collision block $S_{c,1}$		near-collision block $S'_{c,1}$
near-collision block $S_{c,2}$		near-collision block $S'_{c,2}$
$\dots$		$\dots$
near-collision block $S_{c,r}$	$\longleftarrow$collision$\longrightarrow$	near-collision block $S'_{c,r}$
suffix		suffix

Table.

-

prefix, birthday bits, near collision blocks:

- birthday bits: 96, end at the block boundary, they are RSA bits – in the genuine certificate, “tumor” (ignored part by almost all software- marked as a comment extension) – in the rogue certificate

  birthday bits make the difference of intermediate hash values computed for both certificates fall into a *good class*

  birthday paradox makes it possible: we may try many possibilities for tumor

- then apply 3 near-collision blocks of 512-bits. website: we have “consumed” 208 + 96 + 3·512 = 1840 bits of the RSA modulus. Rogue certificate: all bits concerned are in the “tumor”

we can fix remaining bits of RSA key
in arbitrary way, but we need
the matching private key to sign
the certificate request

$B = \boxed{\text{1840 bits already fixed} \quad | \, 111 \ldots 1}$

- after collision bits: 2048-1840 = 208 bits needed to complete the RSA modulus of the webpage – we have to generate an RSA number with the prefix of 1840 bits already fixed?

  – continued so that two prime factors obtained:

→ B denotes the fixed 1840-bit part of the RSA modulus followed by 208 ones

→ select at random 224-bit integer q until $B \bmod q < 2^{208}$, continue until both q and $p = \lfloor B/q \rfloor$ are prime. Then

  – $p \cdot q$ is an RSA number

  – $p \cdot q < B$, $B - p \cdot q = B - q \cdot \lfloor B/q \rfloor < 2^{208}$. Hence $p \cdot q$ has the same 1840 most significant bits as B

→ this RSA number is not secure, but still factorizing it is not feasible and cannot be checked by CA before signing (as the smallest factor is more than 67-digit prime)

→ ... one can create RSA signature for the certificate request

- attack complexity (number of hash block evaluations) for a chosen prefix MD5: 2^{49} at 2007, 2^{39} in 2009, not much motivation for more work - remove MD5 certificates! (For a collision: 2^{16})

  for SHA-1 still 2^{77} in 2012 (for a collision: 2^{65})

- history:

1) $\rightarrow$ attack found

2) $\rightarrow$ real collision computed as a proof-of-concept

3) $\rightarrow$ CA informed and given time to update

4) $\rightarrow$ publication

5) $\rightarrow$ code available

FLAME

- malware discovered 2012, 20MB, sophisticated code, mainly in Middle East, government servers attacked
- draft of the attack:
  - client attempts to resolve a computer name on the network, in particular makes WPAD (Web Proxy Auto-Discovery Protocol) requests
  - Flame claims to be WPAD server, provides wpad.dat configuration file
  - victim that gets wpad.dat sets its proxy server to a Flame computer (later no sniffing necessary!)
  - Windows updates provided by FLAME computer. The updates must be properly signed to be installed!
  - signatures obtained for terminal Services (not for Windows updates!), certificates issued by Microsoft.
  - till 2012 still signatures with MD5 hash

detailed picture of finding hash collision