Applied Cryptography

51484 characters in 8618 words on 1309 lines

Florian Moser

June 16, 2021

1 Introduction

1.1 notation

$ (choose random)
π (random permutation G $\rightarrow$ G)

1.2 secure communication model (informal)

for key K, plaintext p, ciphertext c
c = enc(p, K), p = dec(c, K)

passive adversary
learns all messages exchanged (like c's)
might know context of communication
but does not know key

1.3 kerckhoffs principles

unbreakable (theoretically or practically)
compromise of system details should not decrease security
key memorable without notes
key easily changed
ciphertext transmissible by telegram
encryption apparatus portable & operable by single person
easy (few rules, no mental strain)

modern interpretation
security should relay on key only
specifically, not on system secrecy

"security through obscurity"
systems can and will be reverse engineered (if effort < payoff)
hence system should be secure even if specification known
open specification encourages review & analysis

1.4 shannon's principles

generally accepted design principles for practical ciphers

confusion
ciphertext statistics depends on plaintext statistics
which is too complicated to be exploited by the cryptanalyst

diffusion
each digit of the plaintext and each digit of the secret key
should influence many digits of the ciphertext

1.5 definitions

concrete security
specific about resources of adversary (#queries, time)
$\rightarrow$ for specific key length, #queries, #times, get specific security

asymptotic security
define everything in relation to "security parameter 1^n"
then argue secure if n bigger than some n_0

super-poly set
for 1^n the security parameter
set grows larger than polynomial (like 2^n)

1.6 reduction proofs

show hardness-assumption $\Rightarrow$ scheme-security (like PRF $\Rightarrow$ S secure)
transform to not scheme-security $\Rightarrow$ not hardness assumption

game
assume attacker A breaking scheme S exists
challenger C for hardness assumption exists
define attacker B using A outputs/queries to answer C correctly
B has to simulate perfect environment for A
so A mistakes B with the scheme S challenger

runtime
keep track of queries / other resources consumed by B

overall argumentation
constrain A's success probabilities by calculated probabilities
or other attackers we assume win only with low probability
like Adv(A) < Adv(B) for B negligble as breaking hardness assumption

argue =
let B simulate environment of A
Adv(A) = |Pr[b_d'=0|$b_{d=0}$] - Pr[b_d'=0|$b_{d=1}$]| = |p_0' - p_1'|
Adv(B) = |Pr[b'=0|b=0] - Pr[b'=0|b=1]|
It follows from the construction that p_0 = p_0' and p_1 = p_1'
therefore Adv(B) = Adv(A)

argue $\geq$
let B simulate environment of A
It follows from the construction that B wins at least whenever A wins
therefore Adv(B) $\geq$ Adv(A)

advantage definitions
Adv(A) = 2 $*$ |Pr[Game DDH(A) $\Rightarrow$ true] - 0.5| (for games with true/false)
Adv(B) = |Pr[b' = b] - 0.5| (for bit output which is compared)

1.7 game hoping proofs

define start game G_0 (what to proof)
define end game G_n (with known Adv like hardness assumption)
define games G_i i<n continuously transforming G_0 into G_n
then argue that G_n is negligible / known
then show that each G_i - G_{i+1} is negligible

advantage argumentation
Adv(A) = 2$*$| Pr[G_0] - 0.5 | (by definition)
= 2$*$| Pr[G_0] - Pr[G_1] + Pr[G_1] - (...) - 0.5 | (telescoping sum)
= 2$*$| Pr[G_0] - Pr[G_1] | + 2 $*$ |Pr[G_1] - (...) - 0.5 | (triangle equation)
then show each | Pr[G_0] - Pr[G_1] | is negligible

1.8 nonce-based cryptography

nonce is number-used-once
need only be unique (neither randomness nor secrecy required)
guarantees vanish if uniqueness violated

implementation options
stateful counter (but hard to synchronize state)
randomized (but birthday attack)
transmit explicitly or implicitly (=inferable from counters)

2 probability analysis

2.1 exhaustive key search

for each candidate K, test if pair 1 is valid
with surviving keys, check pair 2, then pair 3, ...

statistical analysis
model encryption E_K as independent random permutation for each K
hence $\Pr_K(C_1 = \mathrm{E}_K(P_1)) = 2^{-n}$ (P_1 mapped to some entry in n)
for k keys, $2^k * 2^{-n} = 2^{k-n}$ survive
for t pairs, 2^{k-tn} survive
hence choose t such that Pr(survive) << 1

complexity
data complexity small (few pairs needed)
computational complexity high, dominated by first step
first step requires 2^k $*$ #pairs tries, afterwards only surviving keys

practicality
works with ciphertext-only attacks if plaintexts are meaningful
meaningful = can decide if plaintext makes sense
exhaustive search is embarrassingly parallelizable

2.2 birthday attack analysis

sample t times from set of size s
all sampled different = 1 $*$ (s-1)/s $*$ ... $*$ (s-t+1)/s

collision probability
1 - (1-1/s)$*$(1-2/s)$*$...$*$(1-(t-1)/s) (bc - all sampled different)
= 1 - exp(log((1-1/s)$*$(1-2/s)$*$...)) (bc exp(log(x)) = x)
= 1 - exp(sum(log((1-j/s)) (bc log(product(x)) = sum(log(x)))
= 1 - exp(-sum(j/s)) (bc log(1-x) = ˜x for small x)
= 1 - exp(- t$*$(t-1)/2s) (bc sum of integers)
> 1 - exp($-t^2$ /2s) (simplify)

evaluations
for t ˜ $s^{0.5}$ ⇒ 0.39
for t ˜ 1.17 $*$ $s^{0.5}$ ⇒ 0.5
for t ˜ 3.03 $*$ $s^{0.5}$ ⇒ 0.99

example for 2^{64} length
low for 2^{30}
raises very rapidly at 2^{32}
almost 1 from 2^{34} until end

hash function implications
n-bit hash function offers only n/2 bits of security
(hence produce collision with $2^{n/2}$ operations + memory)
like require 256 bit output for 128 bit security level

3 one-time pads / perfect security

3.1 perfect security

Pr[P=p|C=c] = Pr[P=p]
posterior of plaintext P, given ciphertext C
equals prior probability of P

3.2 caesars cipher

key determines letter forward shift with wrap-around
if key = 2 then A ⇒ C, Z ⇒ B, ...
$c_i = p_i$ + K mod 26

history
used by the romans

security
26 possible keys (with K=0 that does not hide plain)
leaks length (word length, text length)

frequency analysis
analyse letter frequency and assign key probability
like E which is more frequent than T

3.3 vigenere cipher

letters of key determine alternating forward shift of letter
like multi-key caesars
if key = (1,2) then AAA... ⇒ BCB...
$c_i = p_i + K_j$ mod 26 for j = i mod |K|

history
believed unbreakable for 300+ years

security
26^t possible keys
leaks length (word length, text length)

frequency analysis
for known key length, works same as for caesars
but harder, as less text available
for unknown key length use statistical analysis by kasiski

determine key length
find repeated group of letters
happens when same word is encrypted with same offset
then take lowest common multiplier of all occurrences

3.4 one-time-pad

each letter forward shifted as letter at same position in key
$c_i = p_i + K_i$ mod 26, $p_i = c_i$ - K_i mod 26
for bits, simply use XOR instead of - and +
requires |K| ≥ |P| (impractical)
essentially what all practical schemes try to approximate

security
26^t possible keys
leaks length (word length, text length)
perfect secrecy if K is uniform random and used only once

example
for 7-letter one-time pad & P = {big cats}
all possible responses (cheetah, panther) equally likely

disadvantages
K needs to be as long as message (key space ≥ message space)
K needs to be transmitted to receiver (key management problem)
K used more than once breaks scheme (XOR ciphertexts together)

reusing key attack by NSA
russian agents reused keys when run out of fresh key
NSA used statistical analysis to decrypt plaintexts
⇒ attackers can store ciphertexts for decades!

4 block ciphers

encrypt / decrypt blocks (for example n = 64)
use AES as a rule of thumb

4.1 applications

construction of other block ciphers (like triple DES)
encryption schemes
hash functions
stream ciphers
message authentication codes
pseudorandom bit generators

4.2 perfect security (computational version)

c leaks nothing about p (except previously known)
computationally infeasible to calculate anything useful about p from c

semantic security
for effective A given encryptions of p of its choice
any of A's output can be simulated by S
for S only access to length of c (but not c itself)

IND-CPA security
for effective A given an encryption c of either p_1 or p_2 of its choice
A is unable to distinguish which was encrypted for $|p_1| = |p_2|$

4.3 block ciphers

definition
for key length k, block size n
defines two sets of efficiently computable permutations
E_{K}: $\{0,1\}^n \rightarrow \{0,1\}^n$, D_{K}: $\{0,1\}^n \rightarrow \{0,1\}^n$
such that D_K is an inverse of E_K for all K ∈ $\{0,1\}^k$

alternative definition
let E: K × X → Y be block cipher if
(1) X = Y
(2) for all K ∈ K, E_{K}: X → X is efficiently computable permutation on X

4.4 attacker capabilities

known plaintext attack (KPA)
when adversary observes many (p,c)
like attacker-known IPsec parts (TCP, UDP packet fields)

chosen plaintext attack (CPA)
when adversary chooses many p's and is given c's
like attacker-supplied js encrypting content over TLS

chosen ciphertext attack (CCA)
when adversary chooses many c's and is given p's
like attacker-supplied IPsec packages to produce ICMP errors

4.5 generic attacks

for block cipher to be considered secure, generic attacks must be best attacks

exhaustive key search for key extraction
attacks run under same fixed key K
adversary tries to obtain said key K
adversary capabilities determine strength of notion
no faster method than exhaustive search must be possible

application with alternative targets
might need additional security goals (besides key extractions)

using related keys (motivating related key attacks)
like both K and K XOR R as used as keys
using as key derivator (requires pseudorandom outputs)
like $K_1 = \mathrm{E}_K(nonce_1)$ and $K_2 = \mathrm{E}_K(nonce_2)$
using as perfect security building block (requires provable randomness)
like (q, t, ϵ)-security

randomness
key spaces only motivates 2^k permutations
much smaller than possible $(2^n)!$ possible permutations
prove "enough" randomness ⇒ PRP
might still fulfil randomness security

4.6 pseudo-random permutation (PRP) security

b ← \${0,1}, K ← $\$\{0,1\}^k$ and $\pi \leftarrow \$\text{Perms}[\{0,1\}^n]$
efficient adversary queries oracle with x_i
if b = 0, then oracle responds with $\mathrm{E}_K(x_i)$
else oracle responds with $\pi(x_i)$
adversary decides on b' = {0, 1}
$\mathrm{Adv}_E{}^{PRP}$(A) := 2 |Pr[b' = b] - 0.5|

Game PRP(A, E)
b ← \${0,1}
K ← $\$\{0,1\}^k$
$\pi \leftarrow \$\text{Perms}[\{0,1\}^n]$
b' ← $\mathrm{A}^{F_n}()$
Return b' = b

Oracle $F_{N(x)}$
If b = 0 then y ← E_K(x)
Else y ← π(x)
Return y

strong PRP
if adversary gets access to decryption oracle
hence oracle using D_K(x) and π^{-1}(x)

(q, t, ϵ)-secure as a PRP
if for all adversary A under #time < t, #query < q
advantage $\mathrm{Adv}_E{}^{PRP}$(A) = 2 $*$ |Pr[Game PRP(A, E) ⇒ true] - 1/2|

notes
definition only for uniform random K, b, π
can sample π "as we go"
-0.5 to measure advantage to random sampling
$*$ 2 to normalize result to range 0-1

lazy sampling in oracles
use lazy sampling to avoid having to define full PRP beforehand
receive x_i
if x_j exists for j < i, respond same as in round j
else pick $y_i \leftarrow \$\{0,1\}^n$
if PRP, then ensure y_i does not equal any previous values

4.7 choosing a block cipher

standards
NIST (US), NESSIE (EU), Cryptrec (JP)
also russian, chinese standards

relevant factors
key size (as primary security indicator)
block size (as secondary security indicator)
cryptoanalysis results
standardisation / support in crypto libraries
implementation cost (code/state size)
runtime cost (energy, throughput, , hardware support)
key agility (easy of changing keys)
hardware support (CPU instructions)
implementation security (side channel attacks, ...)

constrained environments
implementation/runtime cost more relevant
want to reduce number of transitions (GE) or at least reuse these
specialized block ciphers need only <1000 GEs (vs >3500 AES GE)

4.8 block cipher constructions

iterate multiple times over simpler, keyed-round function
round keys determined by key schedule (seeded by actual cipher key)
more rounds = stronger algorithms (tradeoff speed & security)

feistel cipher
split block into two halves
apply function to single half, XOR with other
then switch halves and continue
decryption & encryption can use same circuits / code
like DES

substitution-permutation (SP) network
first perform substitution on bits ("confusion")
increases complexity (relation plain/cipher)
then permute / diffuse ("diffusion")
makes each bit dependent on each other bit
hence follow directly shannons's principles
like AES

4.9 DES

exists since more than 40 years, unbroken
constructed together with NSA, which strengthened design
inspired new cryptoanalysis techniques (like linear / differential attacks)

concept
feistel cipher operating on right half
$(L_0 \mid\mid R_0)$ = p
$L_1 = R_0$, $R_1 = L_0$ XOR f(R_0, K)

function f
expands from 32 bits to 48bits (by repeating some bits)
XOR with round key K
substitutes using 8 S-boxes (each mapping 6 to 4 bits)
permutes the 32 bits

architecture
initial permutation IP applied
feistel cipher for 16 rounds
no swap after last round (enables Enc = Dec code)
IP^{-1} is applied

security
too small keys (k=56)
hence exhaustive key search possible
too small block size (64bits)
like sweet32 attack

exhaustive key search
software-only still 100 years bruteforcing
but can use FPGA (few 1000\$) to speed up
professional services available like crash.sh

improvements
double DES not useful ("meet-in-the-middle" attacks)
triple-DES (E_{K_1}, D_{K_2}, E_{K_3}) in use, but quite slow
EDE and EDE_2 variants exist (the latter sets $K_3 = K_1$)

4.10 AES

competition for block cipher to replace DES 1998
requirement to be faster & securer as two-key triple DES
rijndael by belgiens won, pronounced "AES" in 2001
128bits blocks, 128bits keysize (192, 256 also defined)
NSA approved 128bits for secret, 192 for top secret

cryptoanalysis
AES subject to intense analysis (during competition & after)
related key attacks (but impractical)
reduced-round attacks (but >6 of the 10 rounds not anymore)
key recovery attacks (but requires a 2^{126} workload)

side channels
AES needs key-dependent table lookups
remote server & 200M plaintexts by bernstein (2005)
local cache-timing & 800 plaintexts by shamir (2005)

round
state described by 16 bytes
SubBytes substitutes with 8-to-8-bit S-boxes
ShiftRows shifts row 0 by 0 places, row 1 by 1 places, ...
MixColumns multiplies by matrix M (diffusion)
XOR with round key bytes

architecture
initial AddRoundKey (slow; so rekeying painful)
10/12/14 rounds for 128/192/256-bit keys
skip MixColumns in last round

summary
elegant design, good performance (hardware & software)
widely deployed, specialized hardware
no significant direct attacks

implementations may be vulnerable to side-channel attacks

5 symmetric encryption

5.1 symmetric encryption

for m $\in$ M $\subset \{0,1\}^*$, c $\in$ C $\subset \{0,1\}^*$, K = $\{0,1\}^k$
KGen() $\rightarrow$ K, Enc(K, m) $\rightarrow$ c, Dec(K, c) $\rightarrow$ m|error
require (perfect) correctness (for all K (Dec$_K$(Enc$_K$(m)) = m)

notes
Enc non-deterministic, Dec deterministic
M might be restricted to some maximal length, M = $\{0,1\}^{<=L}$
want to minimize c-p $\geq$ 0 (difference through prepending IV, etc)

5.2 modes of operations

to encrypt long messages / continuous stream with block cipher
determine how blocks are composed together
may need pad messages to fill up multiples of block size

5.2.1 electronic code block (ECB)

encrypt block for block
easily parallelizable
messages need to be padded

security
single bit-error in $c_i \Rightarrow p_i$ garbage
but deterministic (given plain always encrypts to same cipher)
hence leaks information about plaintext structure

5.2.2 cipher block chaining (CBC)

XOR plain with previous cipher (or IV) before encryption
IV needs to be uniform-random; sent in plain to decrypter
no parallelization
messages need to be padded

security
single bit-error in $c_i \Rightarrow p_i$ garbage, p_{i+1} same bit-error
IV bit errors $\rightarrow p_1$ same bit-errors

ciphertext-block collisions
if c_i and c_j, then p_i XOR $c_{i-1} = p_j$ XOR c_{j-1}
becomes possible due to birthday attack at $2^{n/2}$

5.2.3 counter mode (CTR)

XOR plain with encrypted counter, increment in mod 2^n for next round
counter can be freely chosen; sent in plain to decrypter
parallelizable
no padding needed (can even truncate last block)
effectively a stream cipher (but dedicated stream designs faster)

security
single bit-error in $c_i \Rightarrow p_i$ same bit-error
if counters repeat, then XOR plains = XOR ciphers
provably secure with game hopping (PRP $\Rightarrow$ PRF $\Rightarrow$ random)
PRP "counters" that structured plain is encrypted (ctr || ctr + 1 || ...)

counter-picking
ctr = 0 & change key each time (but impractical)
ctr = 0 & increment globally (but require state synchronization)
ctr = random (but need good source, avoid overlapping)
ctr = time (but needs conversion, time shift, parallelization)

real-world counter-picking
nonce supplied by application, concatenated with internal counter
like ctr = nonce || 0000, ctr + 1 = nonce || 0001
enforces max message length (depending on internal counter length)

encryption = decryption
can use same algorithm for both
E_K is used only in one-way, hence does not need to be invertible
can use pseudorandom function (instead of pseudorandom permutation)

5.2.4 other modes

GCM (additionally authenticates)
CFB (creates stream cipher; self-synchronizing)
OFB (creates stream cipher; not self-synchronizing)
IGE (used in telegram; not much analyzed)
most common are CBC, CTR, GCM
https://csrc.nist.gov/projects/block-cipher-techniques/bcm/current-modes

stream cipher
dedicated designs for arbitrary length encryptions
might also be supported by hardware

5.3 indistinguishable under CPA (IND-CPA)

challenger C chooses b $\leftarrow$ \${0,1} and K $\leftarrow$ \$KGen
adversary A can query equal length (m_0, m_1)
left-or-right (LoR) encryption oracle responds with c $\leftarrow$ \$Enc$_K(m_b)$
after q queries, A decides on b' = {0,1}
Adv$_{SE}{}^{IND-CPA}$(A) := 2|Pr[b'=b] - 0.5|

Game IND-CPA(A, SE)
b $\leftarrow$ \${0,1}
K $\leftarrow$ \$KGen
b' $\leftarrow A^{LoR(.;.)}$ ()
Return (b' = b)

Oracle LoR(m_0, m_1)
c $\leftarrow$\$ Enc$_K(m_b)$
Return c

(q, t, ϵ)-secure symmetric encryption (SE)
if for all adversary A under #time < t, #query < q
Adv$_{SE}{}^{IND-CPA}$(A) = 2 $*$ |Pr[Game IND-CPA(A, SE) $\Rightarrow$ true] - 1/2|

captured attack notions
message recovery attacks (given c, get m)
key recovery attacks (given (c,m), get K)
(as the attacker can be converted in IND-CPA adversary)

deterministic schemes
generic game automatically breaks IND-CPA
A sends (m_0, m_0) to LoR, gets c
A sends (m_0, m_1) to LoR, gets c'
if c = c' then b = 0, else b = 1

limitations
requires equal length messages (different lengths unprotected)
ignores integrity
ignores chosen ciphertext attacks
ignores side-channel leakage, implementation vulnerabilities

5.4 game hopping lemmas

advantage rewriting lemma
2 |Pr[b'=b] - 0.5| = |Pr[b'=1|b=1] - Pr[b'=1|b=0]|
single output enough to estimate probability

proof advantage rewriting lemma
Pr[b'=b] - 0.5
= Pr[b'=b|b=1]Pr[b=1] + ... - 0.5 (bayesian expand)
= Pr[b'=b|b=1]$*$0.5 + ... - 0.5 (bc Pr[b=0] = 0.5)
= 0.5$*$(Pr[b'=1|b=1] + Pr[b'=0|b=0]) - 1) (evaluate b, extract 0.5)
= 0.5$*$(Pr[b'=1|b=1] - (1-Pr[b'=0|b=0])) (rewriting -1)
= 0.5$*$(Pr[b'=1|b=1] - Pr[b'=1|b=0]) (invert 1-)

difference lemma
given that events W_1 ˆ ¬ Z iff W_2 ˆ ¬ Z
then |Pr[W_2] - Pr[W_1]| $\leq$ Pr[Z]
useful for game hopping with rare bad event Z
intuitively assume algorithms run with same randomness for W_1 and W_2
argue it holds over full probability space

proof difference lemma
|Pr[W_2] - Pr[W_1]|
= |Pr[W_2 ˆ Z] + Pr[W_2 ˆ ¬ Z] - (Pr[W_1 ˆ Z] + Pr[W_1 ˆ ¬ Z])| (expand)
= |Pr[W_2 ˆ Z] - Pr[W_1 ˆ Z]| (precondition)
$\leq$ Pr[Z] (bc both expressions lie between 0 and Z)

5.5 PRP-PRF switching lemma

any PRF (pseudo-random function) is also a PRP
for block cipher E, adversary A with queries q
|Adv$_E{}^{PRP}$(A) - Adv$_E{}^{PRF}$(A)| $\leq q^2$ / 2^{n+1}
in general, q is small and n large, hence probability small

PRF-security
defined same as PRP, except \$Funcs[$\{0,1\}^n$, $\{0,1\}^n$] used
Func includes all PRP, and additionally all functions
Func may duplicate output values (no bijection requirement)
Adv$_E{}^{PRF}$(A) := 2 |Pr[b' = b] - 0.5|

proof games
run A on three games G_0, G_1, G_2 with different oracles on f
G_0 uses f = E$_K$($*$)

G_1 uses f ← \$Perms[$\{0,1\}^n$]
G_2 uses f ← \$Funcs[$\{0,1\}^n$]
p_i = Pr[W_i] for W_i event that A outputs b' = 1
note that W_i independent of success of A

adversary probabilities
Adv$_E{}^{PRP}$(A) = 2|Pr[b' = b] - 0.5| (by definition)
= Pr[b'=1|b=1] - Pr[b'=1|b=0] (by advantage rewriting lemma)
= |p_1 - p_0| (by construction of games G_1, G_0)
Adv$_E{}^{PRF}$(A) = 2|Pr[b' = b] - 0.5| (by definition)
= Pr[b'=1|b=1] - Pr[b'=1|b=0] (by advantage rewriting lemma)
= |p_2 - p_0| (by construction of games G_2, G_0)
|Adv$_E{}^{PRP}$(A) - Adv$_E{}^{PRF}$(A)| = | |p_1 - p_0| - |p_2 - p_0| |
≤ |p_2 - p_1| (absolute value case distinction)

|p_2 - p_1|
G_1 and G_2 are identical except duplicate y_i in G_2 (event Z)
for q chosen values, $0.5 * q^2$ pairs
Pr[$y_i = y_j$] = 2^{-n}
Pr[Z] ≤ q^2 / 2^{n+1} (#pairs ∗ collision probability)
(while events not independent, treat them such to overapproximate)
|p_2 - p_1| ≤ Pr[Z] (by difference lemma)

5.6 IND-CPA security for CTR mode

simplified CTR-mode pseudo-code
assume that messages are single block
each encryption chooses uniform-random ctr
1. crt ← \$$\{0,1\}^n$
2. r = E$_K$(ctr)
3. c_0 = m XOR r
4. return (ctr, c_0)

games
instantiate IND-CPA game LoR oracle with pseudo-code from above
G0 uses original pseudo-code from CTR-mode
G1 uses random permutation π instead if E$_K$(ctr)
G2 uses random function f instead of E$_K$(ctr)
G3 uses random value instead of E$_K$(ctr)
in G3, no more advantage (essentially one-time pad encrypted)
let X_i be event that b'=b in game G_i; q_i = Pr[X_i]

advantage
Adv$_{CTR}{}^{IND-CPA}$(A) = 2∗|q_0 - 0.5|
|q_0 - 0.5| = |(q_0 - q_1) + (q_1 - q_2) + (q_2 - q_3) + (q_3 - 0.5)|
≤ |q_0 - q_1| + |q_1 - q_2| + |q_2 - q_3| (by absolute, q_3 = 0.5)

|q_0 - q_1| is small
create B_1 (IND-CPA challenger & PRP adversary)
chooses b ← \${0,1}, runs simplified CTR-mode code
for 2., asks PRP oracle (which uses E_K or π depending on d)
for 3., uses m_0 or m_1 of A depending on b
if A returns b'=b, then returns d' = 1 else d' = 0
q_0 = Pr[b'=b|d=0] (as d=0 is G_0) = Pr[d'=1|d=0] (by d' definition)
q_1 = Pr[b'=b|d=1] (as d=1 is G_1) = Pr[d'=1|d=1] (by d' definition)
|q_1 - q_0| = | Pr[d'=1|d=1] - Pr[d'=1|d=0] | = Adv$_E{}^{PRP}$(B_1)
observe that B_1 running time & #queries equal that of A
Adv$_E{}^{PRP}$(B_1) ≤ max{Adv$_E{}^{PRP}$(D): D in t_A time and q_A queries}

|q_1 - q_2| is small
create B_2 like B_1, but PRP-PRF adversary (hence oracle changes)
q_1 = Pr[b'=b|d=0] (as d=0 is G_1) = Pr[d'=1|d=0] (by d' definition)
q_2 = Pr[b'=b|d=1] (as d=1 is G_2) = Pr[d'=1|d=1] (by d' definition)
|q_2 - q_1| = | Pr[d'=1|d=1] - Pr[d'=1|d=0] | = Adv$^{PRP/PRF}$(B_2)
Adv$^{PRP/PRF}$(B_2) ≤ q^2 / 2^{n+1} (by PRF/PRP switching lemma)

|q_2 - q_3| is small
create B_3 like B_2, but PRF-Rand oracle (evaluates PRF or picks random)
q_2 only different to q_3 iff ctr not all distinct (event Z)
(as PRF evaluates to same value with same input)
|q_3 - q_2| ≤ Pr[Z] (by difference lemma) ≤ q^2 / 2^{n+1}

full advantage
Adv$_{CTR}{}^{IND-CPA}$(A) = 2∗|q_0 - 0.5|
≤ 2∗Adv$_E{}^{PRP}$(B_1) + $2 * q^2$ / 2^n
for E (q, t, ϵ) secure, then CTR is (q, t, $2 * \epsilon$ + $2q^2$ / 2^n)

generalize result
for stateful counters, |q_2 - q_3| is zero (no duplication possible)
hence slightly better result
for multi-block messages, q gets higher (number of total queries)
complex statistical analysis leads to about the same result

6 block mode attacks

6.1 padding oracle attacks

arbitrary length plaintexts have to be padded (except CBC)
after message decryption, have to again remove padding
exploit error behaviour when invalid padding is detected

pad(.)
map of {0,1}∗ → {$\{0,1\}^n$}∗
necessarily expaning
needs to be efficient
may be randomized or deterministic

padding oracle
for attacker-chosen C
oracle decrypts & returns whether padding valid or invalid
hence single bit leaked per ciphertext

remarks
like CCA, hence not covered by IND-CPA

details in practice
timing noise can be filtered statistically
"single shot" still attackable if plaintext stays the same
min/max cipher compatibility by appending/cutting random blocks
endemic in applications

SSL/TLS
attacks in 2003, 2013
CBC-mode with padding particularly vulnerable
integrity via MAC (of ciphertext) required

6.2 simplified TLS padding (CBC-AES) has CCA

adds t+1 copies of byte value t; for 0 ≤ t ≤ n/8
like 0x00 or 0x01 || 0x01 or 0x02 || 0x02 || 0x02
for CBC-AES with n=128, at most 16 bytes appended, at least 1

padding oracle attack
for p_t unknown plaintext, and c_t its cipherblock
c$_{t-1}$ XOR with 0x0... || (d_t = 0x..) for all 256 possibilities of d_t
if oracle accepts, then P_t XOR (0x0... || d_t) = ... || 0x00
(or special case P_t XOR (0x0... || d_t) = ... || 0x01 || 0x01)
hence d_t equals last byte of plain
then proceed byte-by-byte (0x01 || 0x01, then 0x02 || 0x02 || 0x02, ...)
then proceed block-by-block (as can place any cipherblock last)

runtime
in 128 calls for each byte
+ need to disambiguate 0x01 || 0x01 or 0x00
can extend to entire block (from right to left byte)
can extend to entire ciphertext (by placing attacked block last)

history
many similar attacks discovered; in major protocols
conclude that CBC-mode with padding is vulnerable in general
need to detect modified ciphertexts (via integrity, like using MAC)

6.3 CBC-mode with predictable IV

breaks IND-CPA of CBC-mode
discovered in 1995

attack
attacker in IND-CPA attack
queries (P_0, P_1) to get back (C_0 (=IV), C_1 (=ciphertext))
queries (P_0 XOR C_0 XOR $C_{0'}$ (= predicted IV)) twice
iff $C_1 = C_{1'}$, then b=0

attacker requirements
(1) place P_0 XOR C_0 XOR $C_{0'}$ as first block (as IV)
(2) know position of attacked block (to strip previous blocks)
hence also possible with multi-block message

IV-chaining
when IV set to last ciper block generated
then attack possible in real word
like SSL 3.0, TLS 1.0, SSH

6.4 BEAST full plaintext recovery

CBC-mode with predictable IVs + chosen boundary priviledge

part 1 (extract single byte)
assume p_0 - p_{14} is known, want to know p_{15}
assume IV predictable, can decide first block of plaintext

iterate over 256 guesses of p_{15} to recover byte as before

part 2 (byte sliding)
assume chosen boundary privilege
hence can set position of unknown byte relative to CBC boundary
now do part 1, shift, do part 1 again, shift again, ...

browser implementation
https cookie of target site included automatically in requests
js of malicious webpage executes request to target site
can pads requests for part 2, part 1 also possible (but complicated)

impact
fixed in TLS 1.1 (uses random IVs)
but updating hard (old clients)
some switched to RC4 (broken later on too) or sent dummy records

learnings
theoretical attack of 1995 becomes practical in 2011
⇒ attacks only get better with time
attack needs sophisticated JavaScript implementation
⇒ need tools of hacker community to make practical attacks

7 hash functions

arbitrary length to fixed length hash value
based on block ciphers or own design

7.1 introduction

applications
message fingerprinting
signature schemes (hash-then-sign)
message authentication codes
key derivations (raw data → key)
password hashing
commitment schemes

standards
nist (SHA-224 - SHA-512, Keccak)
nessie (SHA-256 - SHA-512, whirlpool)
cryptrec (SHA-256 - SHA-512)

non-crypto hash function
used for hash tables, caching strategies, bloom filters, ...
but unsuitable for crypto purposes
sometimes misused, like WEP using CRC
but CRC is linear in message bits

7.2 formal

definition
H: {0,1}$*$ → $\{0,1\}^n$
called n-bit hash function

random oracle model
given any input, outputs n-bit random string
useful for formal security analysis, but formally unsound

7.3 security goals

primary (easy to hard)
pre-image resistance (given H(m), infeasible to find m)
second pre-image resistance (given m_1, find m_2 for H(m_1) = H(m_2))
collision resistance (find m_1 != m_2 such that H(m_1) = H(m_2))

secondary
near-collision resistance (find m_1 != m_2 such that H(m_1) ~ H(m_2))
partial pre-image resistance 1 (given H(m), find parts of m)
partial pre-image resistance 2 (given H(m), find m' for H(m') prefix)
prefix of length l must be found in much less than 2^l hash evaluations

implications
CR ⇒ sPre (unconditionally, as attacker construction trivial)
sPre ⇒ Pre (when |R| > |D|, as attacker-output m must be different)

7.4 generic attacks

for secure hash functions, generic attacks must be best known attacks

pre-image resistance
given y ∈ $\{0,1\}^n$, H behaving like random function
iterate over 2^n messages until H(m) = y found

second pre-image resistance
given m_1, H behaving like random function
iterate over 2^n m_2 until H(m_2) = H(m_1) found

collision resistance
given H behaving like random function
iterate over $2^{n/2}$ m, some H(m') = H(m)
by birthday theorem, n-bit hash functions offers n/2 bits security

7.5 formalizing collision resistance (CR)

collisions must exist, as input much larger than output space
so an efficient algorithm exists (simply outputs hardcoded collision)
hence definition cannot quantify over all efficient adversaries

(t,ϵ)-CR adversary
for A running in time t, outputting with probability ϵ
m_0 and m_1 such that H(m_0) = H(m_1)

CR ⇒ second pre-image resistance
proof not second-preimage resistance ⇒ not collision resistance
choose m_1, then find m_2 (by not second-preimage resistance)
output m_1, m_2

CR ⇒ pre-image resistance (definition over domain)
choose random x, then construct adversary challenge y = H(x)
require going over domain, else counter examples possible

CR !⇒ pre-image resistance (definition over range)
let G be collision-resistant n-bit hash function
we define H as n+1-bit hash function like
if |m| = n then H(m) = 1 || m
else H(m) = 0 || G(m)
attacker has 50% probability to find pre-image (when bit 0 is 1)

7.6 merkle-damgard construction

construct hash function from compression function
iterative design to process arbitrary length
like MD5, SHA-1, WHIRLPOOL

iterated hashing
let n output length, IV of length n, block length k
assume compression function h: $\{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$
assume padding scheme pad(m) such that result length multiple of k
break up pad(m) into l parts of length k
use h(IV, m_0) = t_1, then h(t_1, m_1), until m_l
last iteration does h(m_l, $[len(Y)]_k$)

security
for t (0 ≤ t < k) minimal such that k divides pad(m)
for $[len(m)]_k$ k-bit representation of length
let pad(m) = m || 1 || 0^t || $[len(m)]_k$
then if h is collision-resistant, so it H

h CR ⇒ H CR
proof that h is collision-resistant ⇒ so is H
assume adversary A breaking H, construct B breaking h
A outputs two colliding (padded) messages X, Y
for X,Y, length u,v; blocks x_i, y_i; chaining values s_i, t_i
if |X| != |Y|, then $t_i = s_i$ although x_u != y_v (by padding)
else then some i for x_i != y_i must exist with $t_{i+1} = s_{i+1}$

length extension attack
for h = H(m) known, can compute H(pad(m) || length(m) || m')
by taking h as chaining value for m'
⇒ merkle-damgard cannot be modelled as random oracle

7.7 construct compression function h

interface requirement h: $\{0,1\}^n$ x $\{0,1\}^k \rightarrow \{0,1\}^n$
security requirements include collision resistance, one-wayness, ...

davies-mayer
uses block cipher E
message input is key
chain variable t is (plaintext) input

analysis davies-mayer
if E ideal cipher ⇒ collision resistant compression function
hash output size = block size
hence need big block size to avoid birthday attacks
message blocks set keys
hence need fast rekeying

7.8 hash functions

MD5 (128 bits, 1991, broken)

SHA-1 (160-bit, 1995, collisions found)
SHA-2 (256bit, ...; 2002, fine)
SHA-3 (public design 2015; backup if SHA-2 breaks)
Whirlpool (512-bit; used in TrueCrypt)

usage
MDx used a lot (fast) but not anymore (trivially broken)
SHA-1 phasing out (IPsec, SSL, SSH); broken with reasonable effort
SHA-2 primarily used (and safe for the foreseeable future)

7.9 SHA-1

merkle-damgard with output size n

compression function
iterate 80 times
uses block cipher in Davis-Meyer mode (k=512, n=160)
round function operating on 5∗32 bit words
some shifts, additions, simple bit operations

timeline
1995 initial analysis for SHA-0 leads to SHA-1
2005 first estimated attack for 2^{69} (but want 2^{80})
until 2015 lots of analysis; reduction to 2^{61}
collisions up to 77 rounds of 80 of compression function
2012 NIST retires SHA-1, but introduces again in 2015
late 2014 SHA-1 certificates penalized by google & others
2015 freestart collision attack (allowed to choose IV)
for 65 CPUs a few times
2017 shattered full collision 2^{63} SHA-1 computations
for 6500 CPU + 100 GPU years (2^{63} computations)
2020 chosen-prefix collision on full SHA-1
for 900 GPUs 2 months ($2^{63.4}$ computations)

phasing out broken algorithms
once deployed hard to phase out
like complex protocols with negotiation capabilities
need backwards compatibility
practitioners require practical demonstrations before being convinced
lack of understanding of how attacks only get worse
confusion about which security properties are actually broken
like MD5 pre-image still difficult

7.10 SHA-2

similar to SHA-1 with (k=512, n=256)
64 rounds, 8∗32 bit words
no attacks known faster than generic birthday attack
but twice as slow as SHA-1

7.11 SHA-3 (Ketchak)

result of 2007 - 2015 competition
will replace SHA-2 if (ever) broken
giant-bit permutation at core (instead of Davis-Meyer)
15% slower than SHA-256

sponge-construction
let outer state R, |R|=r=1088
let inner state C (|C|=c=512 bits)
let F: $\{0,1\}^{r+c} \to \{0,1\}^{r+c}$ bit permutation
pad(m) into m_i blocks of r bits
IV is 0^{c+r}
absorbing phase by R XOR m_i, then F(R || C)
repeat as often as message blocks
squeezing phase by repeatedly taking out R, then F(R || C)
repeat as often as required output size

7.12 password hashing

store passwords hashed so breach has less impact

hash
random, account specific value
long enough (64bits) to prevent collisions for random choice
effectively makes precomputation useless

iterations
slow down password bruteforcing by iterating hashes
iteration method should be badly parallelizable
like argon2, scryt, bcrypt
see https://www.password-hashing.net/

encryption
needs careful selection of method (like adobe ECB issue)
needs key management

8 MAC

8.1 properties

integrity (message not modified)
data origin authentication (sender correct)
to prevent attackers from forging messages

8.2 limitations

cannot detect message deletion, replay, reordering
need sequence numbers or other primitives
cannot detect reflection attacks
need directional indicators (who should receive it)
or key separation (different keys for different purposes)

8.3 definition

KGen: $\{\} \to \{0,1\}^k$ = K
Tag: $\{0,1\}^k \times \{0,1\}^* \to \{0,1\}^t$ called τ
Vfy: $\{0,1\}^k \times \{0,1\}^* \times \{0,1\}^t \to$ {0,1}
correctness requires Vfy(K, m, Tag(K, m)) = 1 for all messages, keys

notes
keys usually uniform random
tag length usually small (96 - 128bit)
Tag, Vfy usually deterministic

deterministic Tag ("standard")
hence unique tag t for each (K,m)
construct generic Vfy as Tag(K,m) == τ

target security
security through unforgeability
hard for adversary to compute valid τ without key K

attacker capabilities
got multiple message/tag pairs
tag(m) $\to \tau$ oracle (to choose messages for MAC)
verify(m, τ) $\to$ true|false oracle (to check if forgeries valid)

trivially broken schemes
τ must depend on every bit of the message
like H(m XOR K) gives same MAC for same key-long message prefix
must be hand to recover K given (m, τ)
like H(m) XOR K is trivial to extract K
length extensions must be impossible
like mercle-damgard construction can easily be forged

8.4 formalising security

oracles
tag oracle (send m, receive τ = Tag(K,m))
verify oracle (send (m,τ), receive {0,1} = Vfy(K,m,τ))

weak unforgeability (WUF)
if adversary queries verify oracle with valid (m∗, τ∗)
for no query to tag oracle m∗

strong unforgeability (SUF)
if adversary queries verify oracle with valid (m∗, τ∗)
for no query to tag oracle m∗ with response τ∗

(q_t, q_v, t, ϵ)-(W/S)UF-CMA
for q_t tag queries, q_v verify queries, running time t
has no success probability than ϵ
then weakly/strongly unforgeable under chosen message attack

SUF-CMA ⇒ WUF-CMA
as any WUF adversary breaks SUF
can construct WUF but not SUF schemes
for deterministic Tag, WUF = SUF

WUF- but not SUF-CMA scheme
idea is to ignore first bit of τ to easily generate τ'
let (KGen, Tag, Vfy) be WUF-CMA secure scheme
Tag'(K, m) = 0 || Tag(K, m) = τ'
Vfy'(K, m, τ') = b || τ && Vfy(K,m,τ)
query for some m, XOR first bit of τ and resubmit
success p = 100% for SUF, WUF still secure

avoiding Vfy oracle
for any (q_t, q_v, t, ϵ)-SUF-CMA attacker using verification oracle

there exists an (q_t, t, ϵ / q_v)-SUF-CMA attacker without it
proof by constructing B consuming original attacker mocking Vfy oracle
Vfy oracle chooses random query to respond 1, else responds 0
leads to success p = ϵ / q_v

8.5 generic attacks

random k-bit key guess (2^{-k})
use some (m, τ) pairs for exhaustive key search
q_v queries to verify oracle with random τ (q_v ˆ 2^{-t})

conclusion
need large enough tags, keys and few queries
as q_v queries are inherently online, can constrain in protocol
like TLS accepting only single q_v

8.6 MAC from PRF (MAC(F))

if F PRF, then MAC SUF-CMA
works on message input domain X (constrained by F)

construction
let F be function $\{0,1\}^k$ × X → $\{0,1\}^t$
KGen: K ←\$ $\{0,1\}^k$
Tag(K, m): τ ← F(K, m)
Vfy(K, m) using "standard" mode as deterministic Tag
note input domain restricted to F input domain X

claim
for (q_t, t, ϵ)-SUF-CMA adversary against MAC(F)
there exists (q', t', ϵ')-PRF adversary against F
for t' ˜ t, q' = q_t + 1, ϵ' = ϵ - $1/2^t$

games
G_0 SUF-CMA game, oracle answers using F(K, m)
W_0 when A outputs (m∗, τ∗) such that τ∗ = F(K, m∗)
hence A breaks SUF-CMA
G_1 SUF-CMA game, oracle answers using random function f
W_1 when A outputs (m∗, τ∗) such that τ∗ = f(m∗)
m∗ of W_0, W_1 must be different from tag queries m_1, ..., m_q

constructing B
let attacker A break SUF-CMA of MAC(F)
let B's challenger execute either F(K,m) (b=0) or f(m) (b=1)
B simulates MAC(F) tag oracle for A
when A outputs (m∗, τ∗), B queries m∗ into τ'
B outputs iff τ' = τ∗ b'=1 else b'=0
(note b' is "inverted"; iff τ' = τ∗ then B detected F(K, m))

proof lemmas
(1) $\text{Adv}_F{}^{PRF}$(B) = |Pr[b'=1|b=1] - Pr[b'=1|b=0]|
= |Pr[τ∗ = f(m∗)|A in G_1] - Pr[τ∗ = F(K, m∗)|A in G_0]| (by game construction)
= |Pr[W_1] - Pr[W_0]| (by W_1, W_0 definition)
(2) Pr[W_1] = $1/2^t$ as A must output τ∗ on fresh input m∗
and output for f is uniform-random of length t

proof
Adv{_MAC(F)}$^{SUF-CMA}$(A) = Pr[W_0]
= |(Pr[W_0] - Pr[W_1]) + Pr[W_1]| (valid as Pr[W_0] ≥ 0)
<= |(Pr[W_0] - Pr[W_1])| + Pr[W_1] (valid as Pr[W_1] >= 0)
= $\text{Adv}_F{}^{PRF}$(B) + $1/2^t$ (by (1) and (2))

8.7 MAC from hashing (HtMAC)

if H CR and MAC SUF-CMA, then HtMAC SUF-CMA
works on message input domain X' (constrained by H)

construction
let MAC = (KGen, Tag, Vfy) with input space X, tag-length t, key-length K
let H: X' → X be hash function
Tag'(K,m): Tag(K, H(m))
Vfy'(K, m, τ): Vfy(K, H(m), τ)

claim
for A (q_t, t, ϵ)-SUF-CMA adversary against HtMAC
$\text{Adv}_{HtMac}{}^{SUF-CMA}$(A) ≤ $\text{Adv}_{MAC}{}^{SUF-CMA}$(B) + $\text{Adv}_H{}^{CR}$(C)
B, C run in similar time t as A, B makes q_t queries

probability claims
let X when A wins SUF-CMA against HtMAC
let Y when H(m∗) = H(m_i) for m∗ != m_i
let Z = X ˆ ¬ Y (A wins without hash collision)
$\text{Adv}_{HtMAC}{}^{SUF-CMA}$(A) = Pr[X]
= Pr[X ˆ ¬ Y] + Pr[X ˆ Y] (taking X apart)
≤ Pr[Z] (by def.) + Pr[Y] (as Pr[X ˆ Y] ≤ Pr[Y])

B game construction
let attacker A break SUF-CMA of HtMAC
let B's challenger execute Tag(K, $m_{i'}$)
B simulates HtMAC tag oracle for A
for query m_i of A, B forwards H(m_i) to its challenger
B relays answers of its challenger without change to A
when A outputs (m∗, τ∗), B wins if Z (no collision)

C game construction
let attacker A break SUF-CMA of HtMAC
C simulates HtMAC tag oracle for A
C chooses some K, evaluates T(K, H(m_i))
when A outputs (m∗, τ∗), C wins if Y (H(m∗)=H(m))

proof notes
C does not care if A wins, hence Y is enough to succeed
advantage of attackers equal to winning probability

8.8 HMAC

build MAC using only hash function
as hash functions are very fast
but hash-then-MAC has offline attack (find collision in hash)

target
transform unkeyed primitive to MAC without CR assumption
could prepend key, but length extension attack
could append key, but vulnerable to offline collision attack)
idea F((K_1, K_2), M) = H(K_2 || H(K_1 || M))

construction
given key K, message m, H using merkle-damgard construction
pick IV (reused over both hashes)
pad key with pad(K) = K || $0x0^*$ to block length
pad messages with pad(m) = m || 0x1 || 0^* || $[length]_{64}$ to block length
let K_1 = pad(K) XOR ipad=0x36..., K_2 = pad(K) XOR opad=0x5C...
h(IV, K_1) = t_1; h(t_1, m_1) = t_2, ... → t_u
h(IV, K_2) = v_1; h(v_1, pad(t_u)) = τ
needs h such that pad(t_u) fits in single block

standard security
keys are derived from single key using XOR (hence not independent)
can prove security for NMAC (same construction, independent K_1, K_2)
need h to behave like pseudo-random function
need ideal cipher model (for any K block-cipher behaves pseudo-random)
under these assumptions HMAC is PRF, therefore SUF-CMA MAC

usage
one of the first MAC functions, widely deployed
used with SHA-1, SHA-256, MD-5
OK with MD5 as security not depending on its (broken) CR
is being replaced by faster designs
still useful for key-derivation (due to PRF property)

IUF-interface
initialize() create internal chaining value K XOR ipad
update(U) to absorb new message bytes (hashing on demand)
finalize() which processes remaining bytes and does outer hash
supports streaming applications; implementation tricky

8.9 nonce-based MAC (NMAC)

add nonce space N compared to standard definition
KGen: {} → $\{0,1\}^k$ = K
Tag: $\{0,1\}^k$ × N × $\{0,1\}^*$ → $\{0,1\}^t$ = τ
Vfy: $\{0,1\}^k$ × N × $\{0,1\}^*$ × $\{0,1\}^t$ → {0,1}
correctness requires Vfy(K, m, N, Tag(K, N, m)) = 1

security game
adversary must not query same nonce twice
wins if A outputs m∗, τ∗ for Vfy(K, N∗, m∗, τ∗)
for (N∗, m∗, τ∗) distinct from (N_i, m_i, τ_i)

(q_t,t,ϵ)-SUF-CMA-secure
$\text{Adv}_{NMAC}{}^{SUF-CMA}$(A) < ϵ for t time, q_t tag queries
for forgery in unforgeability game

8.10 universal hash functions (UHF)

useful for (very fast) compression
but does not hide input/key (hence recoverable from hash)
in real-world usecase, hide output by XORing with PRG or similar

$\epsilon - UHF$
when $\text{Adv}_H{}^{UHF}(A) \leq \epsilon$ in universal hash function security game
A plays against challenger with K ←$ K
A outputs m_0, m_1 for m_0 != m_1 and H(K, m_0) = H(K, m_1)
note that A has no access to any pairs beforehand
H(K, m) called keyed hash function (& its output called digest)

$\epsilon - UHF$ alternative definition
A of $\epsilon - UHF$ is unbounded, hence can reformulate
Pr[H(K, m_0) = H(K, m_1)] $\leq \epsilon$
for random K, any m_0 != m_1

8.11 H_{poly} (UHF from polynomials)

for F finite field (like mod p; GF(2^n))
let K = T = F (hence keys, tags = F)
let M = $(F)^{<=l}$ (hence messages vectors of at most length l)
H(K, (a_1, ..., a_v)) = K^v + $a_{1*K^{v-1}}$ + ... ∈ F
write as a(K), as a(X) degree v polynomial from F
fast evaluation using finite field operations & horners rule

H_{poly} is $\epsilon - UHF$
assume a, b distinct
then (a-b)(K) degree > 0 (due to K^v) and degree ≤ l (as |a|, |b| ≤ l)
Pr[(a-b)(K) = 0] ≤ l/|F| (as at most l roots, |F| options for (a - b))
hence Pr[a(K) = b(K)] ≤ l/|F| (equivalent to $\epsilon - UHF$ definition)

PRF from PRF + UHF (UHFtPRF)
for H $\epsilon - UHF$, F PRF
F'((K_1, K_2), m) = $F(K_2$, H(K_1, m)
$Adv_{F'}{}^{PRF(A)} \leq \text{Adv}_F{}^{PRF}(B) + 0.5 * q^2 * \epsilon$
we want to get rid of $0.5 * q^2$ (as its not tight)

8.12 ϵ difference UHF ($\epsilon - DUHF$)

for keyed hash function keyspace K, message space M, digest space T
for T equipped with group operation + (and inverse -)
challenger picks K ←$ K
A outputs m_0, m_1 and δ such that H(K, m_0) - H(K, m_1) = δ
generalizes hash functions (as diff no longer has to be 0)

typical digest spaces & operations
group Z_N (integers mod N) with + as addition mod N
group $\{0,1\}^n$ for some bit length n with + as XOR
XOR group also called $\epsilon - XOR - universality$

8.13 H_{xpoly} (DUHF from polynomials)

for F finite field (like mod p; GF(2^n))
let K = T = F (hence keys, tags = F)
let M = $(F)^{<=l}$ (hence messages vectors of at most length l)
H(K, (a_1, ..., a_v)) = K^{v+1} + a_{1*K^v} + ... ∈ F
= K ∗ H_{poly}(K, (a_1, ..., a_v))

H_{xpoly} is $\epsilon - DUHF$
assume a, b distinct
then (a-b)(K) polynomial degree > 0 and ≤ l+1
Pr[(a-b)(K) = 0] ≤ l+1/|F| (as at most l+1 roots, |F| options for a and b)
hence Pr[K∗(a(K) - b(K)) = δ] ≤ l+1/|F| (equivalent to $\epsilon - DUHF$ definition)

8.14 Carter-Wegman MAC (CW-MAC(F, H))

builds up MAC out of $\epsilon - DUHF$ H and PRF F
used in AES-GCM

construction
let K_1, K_2 keys, N nonce
τ = H(K_1, m) XOR F(K_2, N)
like one-time pad encryption for output of H
results in tags of size |N| + |τ|

SUF-CMA security
for H $\epsilon - DUHF$ and F PRF; both with (T, +) target group
Adv{_CV-MAC(F,H)}$^{SUF-CMA}$(A) ≤ $\text{Adv}_F{}^{PRF}$(B) + ϵ + 1 / |T|
time of B roughly the same, q_t queries to oracle

proof sketch
replace F with random function (valid as nonces differ)
case A outputs (N∗, m∗, τ∗) for N∗ new
then τ∗ = H(K_1, m∗) + f(N∗)
as f(N∗) uniform random, A succeeds with p = $1/|T_H|$
case A outputs (N∗, m∗, τ∗) for N∗ used in previous tag query
then τ∗ = H(K_1, m∗) + f(N) and τ = H(K_1, m) + f(N)
with τ∗ - τ can build $\epsilon - DUHF$ adversary

GMAC algorithm
for F, uses AES (applying PRP-PRF switching lemma)
for H, use H_{xpoly} over GF(2^{128}) for maximum length
special instructions on Intel and AMD chips

Bernstein Poly1305-AES
adds efficiency tweaks using F = GP(p) for p = 2^{130} - 5
may exploits floating point arithmetic

8.15 other constructions

CBC-MAC
with IV = 0, do CBC-encryption, return last cipherblock as τ
SUF-CMA secure if IV constant & fixed-length messages (#blocks equal)
SUF-CMA secure if message length prepended & padding sensible

9 authenticated encryption

9.1 introduction

security goals are confidentiality and integrity
adversary owns network (delete, reorder, modify, ...)
adversary can mount chosen plaintext and chosen ciphertext attacks

non-mallable motivation
IND-CPA does not prevent adversarial bit-flipping attacks
as seen for CBC-mode, CTR-mode
all bitstrings of correct length are valid and will decrypt

9.2 security definitions

integrity of ciphertexts (INT-CTXT)
adversary has access to encryption oracle (m ⇒ $Enc_{K(m)}$)
then can query try oracle a single time
adversary wins if c∗ new and decryption of c∗ succeeds

integrity of plaintexts (INT-PTXT)
same game as INT-CTXT
additionally require the decrypted m∗ has not been queried

INT-CTXT ⇒ INT-PTXT
follows from definitions (any INT-PTXT adversary breaks INT-CTXT)
& correctness of decryption (different plain must mean different cipher)

(q_e, t, ϵ)-secure
for attackers querying encryption oracle q_e times
running in time t
succeeding with probability lower than ϵ

multi-try versions
equivalent definition, q_{try} ∗ advantage

9.3 authenticated encryption (AE-security)

IND-CPA (cannot differentiate encryptions)
INT-CTXT (cannot forge new ciphertexts)
adversary has access to an encryption oracle

9.4 IND-CCA

security game
adversary has decryption oracle & LoR encryption oracle
(for any c returned by encryption oracle, cannot query decryption oracle)
$\text{Adv}_{SE}{}^{IND_{CCA}}$ = 2 ∗ |Pr(b'=b) - 0.5|

AE-security ⇒ IND-CCA security
assume IND-CCA adversary against AE-secure scheme
case adversary comes up with c∗
but this breaks INT-CTXT security
case adversary does not come up with c∗
but then equal game as IND-CPA (as decryption oracle unused)

proof
let A be IND-CCA adversary
event X if A wins (b'=b)
event Y if A queries valid c∗, for c∗ not from encryption oracle
event Z = X ˆ ¬ Y
Pr[X] ≤ Pr[Z] + Pr[Y]
$\text{Adv}_{SE}{}^{IND-CCA}(A) \leq \text{Adv}_{SE}{}^{IND-CPA}(B) + 2q_d *$ $\text{Adv}_{SE}{}^{INT-CTXT}(C)$

construct B (IND-CPA)
handles event Z

B simulates IND-CCA environment for A
B relays encryption oracle queries to own challenger
correct by construction
B responds with bottom to any decryption oracle (event Z)
correct bc ¬ Y implies A's queries are all wrong
B outputs b' = b, wins whenever A wins

construct C (INT-CTXT)
handles event Pr[Y]
C chooses b ← ${0,1}, j ←$ {1, ..., q_d}
C simulates encryption oracle by forwarding m_b to own challenger
correct as same bit with same distribution used
C simulates decryption oracle by returning bottom if not i = j
else relays c to own try(c∗) challenger
succeeds assuming j picked correct (=first instance where Y)
up until Y occurs, A's queries wrong hence returning bottom is OK
C wins if challenger accepts (terminates after try(c) query)
C selects correct j with p $\geq 1/q_d$

9.5 symmetric encryptions security notions

AE (IND-CPA + INT-CTXT) implies all other useful notions
AE ⇒ IND-CCA ⇒ IND-CPA (oracles get stronger)
AE ⇒ IND-CPA + INT-PTXT ⇒ IND-PTXT

counter examples reverse
IND-CCA !⇒ AE (examples in exercises)
IND-CPA !⇒ IND-CCA (counter mode)
INT-PTXT !⇒ IND-CPA (MAC which does not encrypt content)
IND-CPA + INT-PTXT !⇒ AE

other proofs
INT-CCA + INT-PTXT ⇒ AE (not obvious, but proven)

9.6 limitations

does not prevent reordering / deletion of ciphertexts
need integrity protected associated data (use AEAD)

AE implies IND-CCA proof assumptions
assumes only single error message is thrown
but in practice likely multiples (padding / mac / decryption error)
then proof breaks down, but gap closable (paper 2012)

9.7 generic compositions for AE

Encrypt-and-MAC (E&M)
c ← Enc_{KE}(m), τ ← Tag_{KM}(m), output c || τ
but in practice might use m before checking mac

MAC-then-Encrypt (MtE)
τ ← Tag_{KM}(m), output c = Enc_{KE}(m || τ)
but if MAC not randomized, leaks information about plaintext
used in SSL and TLS < 1.3

Encrypt-then-MAC (EtM)
c ← Enc_{KE}(m), τ ← Tag_{KM}(c), output c || τ
reduces temptation to use m before checking τ
MAC needs to cover whole ciphertext, including IV
used in IPsec ESP

9.8 security proofs

EtM gives AE security
for IND-CPA encryption and SUF-CMA MAC scheme AE secure
MAC does not leak information as operates on IND-CPAed ciphertext

E&M not secure w/ deterministic MAC
assume PRF used as MAC (or any deterministic MAC)
query (m_0, m_0) and (m_0, m_1) in IND-CPA security game
if MAC(c_0) = MAC(c_1), then b=0, else b=1
hence broken IND-CPA with two queries

MtE not secure
use SUF-CMA MAC (like HMAC)
use IND-CPA encryption scheme (like CBC-mode)
c = Enc(padding(m || Tag_{KM}(m)))
decryption involves multiple steps; each with its own errors
if error messages distinguishable, can do padding oracle attack
can be made secure, but unsafe in general ⇒ try to avoid

9.9 AE with associated data (AEAD)

have associated data AD that needs to be in plaintext
like ESP header of IPSec

properties
confidentiality for payload m
integrity for combined AD and ciphertext of m

definition
KGen selects uniform random K
Enc(K, AD, m) → c
Dec(K, AD, c) → m | bottom
require correctness
AD sent along / receiver reconstructs

using EtM
c = Enc(K, m), τ = MAC(len(AD) || AD || c)
length of AD prevents miss-parsing (like moving AD into cipher)

9.10 nonce-based AEAD

nonces
easier to provide as good source of randomness
for example, some protocols already keep sequence counters

definition
KGen selects uniform random K
Enc(K, N, AD, m) → c
Dec(K, N, AD, c) → m | bottom
require correctness
AD & N sent along / receiver reconstructs
for N, typically use a synchronized counter
like TLS sequence numbers

security
IND-CPA queries include N like (N, AD, (m_0, m_1))
N must never repeat over different queries
INT-CTXT wins if fresh (N∗, AD∗, c∗) submitted to single try()
fresh means no query exists with (N∗, AD∗, ()) → c∗
can generalize to multiple try() with factor q_{try}

basic secure channel
assume client A sends encryptions to server B with preagreed key
c_0 = Enc(K, N=0, AD_0, m_0), increment N with each message
deletion/reordering detected by B as MAC validation fails
truncation undetected (but could add end message, ACKs)

9.11 further constructions

EtM to AEAD
c ← Enc_{KE}(M), τ ← Tag_{KM}(len(A) || AD || c)
can use nonce-based MAC and Enc (using same nonce)

9.12 AES-GCM

using nonce-based CTR-mode AES, CW-MAC
nonces can be arbitrary length, 96 bits typically used
maximum message length is 2^{32} AES blocks

MAC
using CW-MAC(H, F) construction
H = $H_{xpoly}\ \epsilon - DUHF$ over GF(2^{128})
F = AES
truncation is allowed

encryption(K, N, m)
ctr = N || 0^{31} || 1 (for |N| = 96)
c = AES-CTR-Enc(K, ctr, m)
c' = AD || c || $len(AD)_{64}$ || $len(c)_{64}$
τ = CW-MAC(K_H, K_{PRF}, N || 0^{32}, c')
for K_H = AES(K, 0^{128}), K_{PRF} = K

advantages AES-GCM
almost as fast as CTR-mode (due to fast MAC)
uses block cipher only "forwards" (only Enc)
streaming computation possible
security proof only assumed AES pseudo-randomness
patent-free, clearly specified, widely used (IPsec, TLS)

insecurity under nonce reuse
can recover MAC key & CTR mode fails (XOR ciphers = XOR plains)
then can forge arbitrary packets for specific nonce