Applied Cryptography - Part 2

71147 characters in 12042 words on 1825 lines

Florian Moser

June 16, 2021

1 searchable encryption

want to outsource storage without leaking information
still want to (efficiently) query it

1.1 model

database representation
collection of documents each with own id
search index which maps ids to keywords

search indexes
direct index (doc_{id} ⇒ keywords)
inverse index (keyword ⇒ doc_{ids})

honest-but-curious adversary
server follows designed protocols
but tries to infer as much about data as possible
stronger models exist

snapshot adversary
sees server state at specific point in time
too weak for real databases (due to caches)

1.2 abstract protocol

setup
client generates encrypted database & search index
client sends data to server

search
client sends search token to server
server uses token to process encrypted search index
server returns the result

update
client sends update token to server
server uses token to process encrypted database & search index
server returns success or failure

1.3 goals

security
confidentiality of documents/query
like against honest-but-curious attacker

efficiency
minimal storage / computation requirements at client/server
like low bandwidth, few interactions for single query

functionality
type of queries that are supported
like single keyword, boolean, AND/OR, ...

1.4 default everything

encrypt documents & search index symmetrically
upload as large blob to server
like using AES-GCM

query
download & process locally (but inefficient)
share key with cloud (but insecure)

1.5 PRF construction

client chooses key K for PRF
encrypts keywords in inverse index with PRF(K, keyword)
to query, sends PRF(K, keyword) to the server

leakage setup
number of keywords & documents
frequency of keywords (how often a specific keyword appears)
co-occurrence information for keywords (documents w/ same keyword)

leakage searches
result pattern (queries document result)
query equality pattern (queries over same keyword)
query intersection pattern (common documents over different queries)

1.6 PRF construction 2

client chooses key K for PRF
for each keyword w, get $K_1 \mid\mid K_2 = F_{K(w)}$
set keys of indirect index to K_1
encrypt values under id XOR F_{K_2}(cnt) (for some counter ctr)

improvement
setup no longer leaks co-occurrence (due to PRF)

1.7 key-value store

like PRF-construction 2
use key-value store which hides #values per key for indirect index
client supplies #values of key in queries

example
index "Alice → 1, 5, 6"
encrypts to 3 pairs like
($F_{K1(1)}$, 1 XOR $F_{K2(1)}$), ($F_{K1(2)}$, 5 XOR $F_{K2(2)}$), ($F_{K1(3)}$, 6 XOR $F_{K2(3)}$)

querying
send K1 (key), K2 (to decrypt document ids) and count c
server returns the c queried documents

key-value store implementation options
use key as starting address, take |value| = count
use key as PRF start, repeat count times to get value addresses

improvement
at setup, only learn #documents (but can add dummy documents)
at query time, learn keyword frequency / #keywords

1.8 formally establishing leakage profile

state claim about setup & search
using leakage information, formulate simulator
if adversary cannot distinguish simulator & real world
then scheme is considered secure

(t, t', q, ϵ)-secure with respect to L
iff every adversary A running in time t with q queries
there exists a simulator given L = (L_{setup}, L_{search}) in time t'
succeeds with Pr[b=b'] - 0.5| $\leq \epsilon$

1.9 analysing leakage in searchable encryption

query leakage
for each query, learn how many documents returned
if known how often keyword occur in documents
can infer with high probability which keyword was queried

1.10 extensions

update tokens
leakage analysis much more difficult
require forward & backward privacy

more advanced queries
want OR / AND queries
want range queries
leakage difficult to limit

2 public key encryption (PKE)

different keys for encryption / decryption also called asymmetric encryption

2.1 application

asymmetric cryptography more expensive than symmetric crypto at same security level

hybrid encryption
encrypt message with symmetric key
then encrypt symmetric key using public key cryptography
combines public key advantages with speed of symmetric keys

distribution of authentic public keys
the hard problem of public key encryption
solved using a public key infrastructure (which has its own problems)
like symmetric key distribution, but without having to keep keys secret

2.2 definition PKE

(sk, pk) ← KGen for sk secret key, pk public key
c ← Enc(pk, m) (usually randomized)
m|bottom ← Dec(sk, c)
correctness requires $\text{Dec}_{sk}(\text{Enc}_{pk}(\text{m})) = \text{m}$ for all KGen outputs

2.3 formalizing security

nothing about plaintext leaks to the adversary

IND-CCA
challenger chooses b ← {0,1} and (sk, pk) ← KGen
adversary is given pk
then can query encryption / decryption oracle
may not query decryption oracle with c received from encryption oracle
outputs bit b' deciding on b

(q_e,q_d,t,ϵ)-IND-CCA secure
for q_e encryption queries, q_d decryption queries, t time
$\text{Adv}_{PKE}{}^{IND-CCA}(\text{A}) = 2 * |\text{Pr[b=b']} - 0.5| < \epsilon$

IND-CPA
like IND-CCA, but without decryption oracle
usually build IND-CPA secure system first
then add mechanism to get IND-CCA

IND-CCA ⇒ IND-CPA
any IND-CPA adversary breaks IND-CCA
simply does not use decryption oracle

fixed q_e
q_e usually fixed to 1
for $q_e > 1$, get security loss in same factor

2.4 KEM/DEM-construction

key encapsulation mechanism (KEM)
out of public key, generates symmetric key K and its encryption c
same key K can be recovered using c and the private key
with algorithms (KEM.KGen, KEM.Enc, KEM.Dec)

data encapsulation mechanism (DEM)
symmetric encryption mechanism using K of KEM
used to encrypt / decrypt actual payload
with algorithms (DEM.KGen, DEM.Enc, DEM.Dec)

construction
PKE.KGen runs (sk, pk) ← KEM.KGen
outputs (sk, pk)
PKE.Enc(pk, m) runs (c_0, K) ← KEM.Encap(pk)
outputs (c_0, c_1 ← DEM.Enc(K, m))
PKE.Dec(sk, (c_0, c_1)) recovers K ← KEM.Decap(sk, c_0)
returns m ← DEM.Dec(K, c_1)

space requirements
KEM.K = DEM.K (same K space)
PKE.M = DEM.M (PKE messages space given by DEM)
PKE.C = KEM.C × DEM.C

2.5 KEM-security

definition
(sk, pk) ← KGen for sk secret key, pk public key
(c, K) ← Encap(pk) (usually randomized)
K|bottom ← Decap(sk, c)
correctness when (c,K) ← Encap(pk) implies K ← Decap(sk, c)

IND-CPA
challenger chooses b ← {0,1} and (sk, pk) ← KGen
challenger calculates (c∗, K_0) ← Encap(pk)
challenger chooses K_1 ←$ K
adversary is given pk, c∗, K_b
outputs bit b' deciding on b

IND-CCA
before deciding, can additionally query decryption oracle
receives result of Decap (K or bottom)
only for c != c∗

(q_d,t,ϵ)-IND-CCA secure
for q_d decryption queries, t time
$\text{Adv}_{KEM}{}^{IND-CCA}(\text{A}) = 2 * |\text{Pr[b=b']} - 0.5| < \epsilon$

2.6 IND-CCA security of KEM/DEM construction

for IND-CCA secure KEM and IND-CCA secure DEM
$\text{Adv}_{PKE}{}^{IND-CCA}(\text{A}) \leq 2 * \text{Adv}_{KEM}{}^{IND-CCA}(\text{B}) + \text{Adv}_{DEM}{}^{IND-CCA}(\text{C})$
for A $q_e = 1$ and both B and C make same number of q_d queries as A

games
assume IND-CCA attacker on PKE with enc/dec oracle
G_0 is original challenger (PKE.Enc, PKE.Dec functions as defined)
G_1 is KEM challenger b=0 (enc uses K_0, dec uses c_0 or queries)
G_2 is KEM challenger b=1 (enc uses K_1, dec uses c_0 or queries)
G_3 is DEM challenger (encap/decap all done inside B)
let X_i be event that b'=b in game $G_{i;}$ $q_i = \text{Pr}[X_i]$

advantage
$\text{Adv}_{CTR}{}^{IND-CPA}(\text{A}) = 2*|q_0 - 0.5|$
$|q_0 - 0.5| = |(q_0 - q_1) + (q_1 - q_2) + (q_2 - q_3) + (q_3 - 0.5)|$
$\leq |(q_0 - q_1)| + |(q_1 - q_2)| + |(q_2 - q_3)| + |(q_3 - 0.5)|$
$\leq 0 + \text{Adv}_{KEM}{}^{IND-CCA}(\text{B}) + 0 + 0.5 * Adv_{DEM}\{^{IND-CCA)(C)}$
as $G_0 \to G_1$ and $G_2 \to G_3$ are only syntactic changes
as $|q_1 - q_2|$ allows us to construct $\text{Adv}_{KEM}{}^{IND-CCA}(\text{B})$
as G_3 is exactly IND-CCA DEM challenger
G_3 result implies only one-time security of DEM is required

$|q_1$ - $q_2|$
given IND-CCA PKE adversary A
plays against IND-CCA KEM challenger C having hidden bit b
choose b ← ${0,1}, receive pk, c∗, K_d of C
answer encryption queries (m_0, m_1) with Enc(K_d, m_b)
answer decryption queries (c_0, c_1) iff c_0 == c∗ with Dec(K_d, c_1)
else getting K' with dec(c_0) from C, then Dec(K', c_1)
if A returns b'=b, then returns d'=1 else d'=0
q_1 = Pr[b'=b|d=0] (as d=0 is G_0) = Pr[d'=1|d=0] (by d' definition)
q_2 = Pr[b'=b|d=1] (as d=1 is G_1) = Pr[d'=1|d=1] (by d' definition)
$|q_2 - q_1|$ = |Pr[d'=1|d=1] - Pr[d'=1|d=0]| = $\text{Adv}_{KEM}{}^{IND-CCA}(\text{B})$
observe that B running time & #queries equal that of A

generalize result to any q_e
possible; results in q_e factor in bounds
still only one-time query of DEM is required

3 number theory

3.1 notation & terminology

N for non-negative integers
a mod b = c for c reminder of a/b
a divides (factor of, |) b iff a/b = 0
Z_{n*} for totatives of n (numbers < n & not dividing n)

3.2 greatest common divisor (gcd)

largest number factoring two numbers
like gcd(3,5) = 1, gcd(4,6) = 2

euclidian algorithm (gcd)
gcd(a,0) → a
gcd(a,b) → gcd(b, a mod b)
gcd(9,6) → gcd(6, 9 mod 6) → gcd(3, 6 mod 3) → 3

3.3 extended euclidian algorithm (EEA)

calculates bézout's identity a∗x + b∗y = gcd(x,y)

build up table
$x = 1 * y + d_1 \Rightarrow d_1 = x - 1 * y$
$y = t_1 * d_1 + d_2 \Rightarrow d_2 = y - t_1 * d_1$
$d_1 = t_2 * d_2 + d_3 \Rightarrow d_3 = d_1 - t_2 * d_2$
...

start with given x = y + rest
maximize t_i, then calculate $d_{\{i+1\}}$ trivially
right column helps with reconstruction later
proceed to next row shifting all to the left

gcd(19, 12) build up
19 = 1 * 12 + 7 ⇒ 7 = 19 - 1 * 12
12 = 1 * 7 + 5 ⇒ 5 = 12 - 1 * 7
7 = 1 * 5 + 2 ⇒ 2 = 7 - 1 * 5
5 = 2 * 2 + 1 ⇒ 1 = 5 - 2 * 2
2 = 1 * 1 (done; gcd(19, 12) = 1)

reconstruct linear combination
1 = d_2 - $t_3 * d_3$
1 = d_2 - $t_3 * (d_1$ - $t_2 * d_2)$
= $t_3 * t_2 * d_2$ - $t_3 * d_1$
1 = (replacing d_2, ...)
...
start with last row, left column of table
insert next upper row, then multiply out

gcd(19, 12) reconstruction
1 = 5 - 2 * 2
1 = 5 - 2 * (7 - 1 * 5) = 3 * 5 - 2 * 7
1 = 3 * (12 - 1 * 7) - 2 * 7 = 3 * 12 - 5 * 7
1 = 3 * 12 - 5 * (19 - 1 * 12) = 8 * 12 - 5 * 19

alternative solution
can use table-based approach
i	q_i	r_i	t_i	s_i
0	-	a	0	1
1	?	b	1	0
determine q_i as max within a - $q_i * b$
then calculate r_i, t_i, s_i next = previous - $q_{i*current}$
finished if $r_k = 0$ for some k
solution in t_{k-1}, s_{k-1}

application
for EEA(a, p) with some number a, prime p
results in a*s + p*t = 1 ⇒ a*s = 1 mod p
hence useful to find inverse of a in mod p

3.4 chinese reminder theorem

for m_i relatively pairwise
system of congruences x = a_i mod m_i
has solution x_0 = sum n_{i*x_i} for n_i = (mul m_i) / m_i

decompose equations
each m_i has to be relatively prime
x = 3 mod 10 → x = 3 mod 5 and x = 3 mod 2

find x_i
calculate m (multiplying all m_i)
calculate n_i (m / m_i) & apply modulo
determine x_i such that equation works

build sum
sum up all $x_i * n_i$ to get x_0
final result is x = x_0 mod m

example
2 mod 3, 3 mod 4, 4 mod 5
check 3,4,5 relatively prime → yes, hence decomposing
m = 3 * 4 * 5 = 60
n_1 = 4*5, n_2 = 3*5, n_3 = 3*4
$20 * x_1$ = 2 mod 3 <=> $2 * x_1$ = 2 mod 3 ⇒ $x_1 = 1$
$15 * x_2$ = 3 mod 4 <=> $3 * x_2$ = 3 mod 4 ⇒ $x_2 = 1$
$12 * x_3$ = 2 mod 5 <=> $2 * x_3$ = 4 mod 5 ⇒ $x_3 = 2$
x = 20*1 + 15*1 + 12*2 = 59 mod 60

3.5 primes

p prime iff factors only with itself and 1 (p > 1)
coprime (=relatively prime, ⊥) iff gcd(a, b) = 1

multiset of primes I(n)
n determined by product of multiset of primes I(n)
a,b divisible iff I(a) ⊆ I(b)
a,b coprime iff I(a) ∩ I(b) = ∅
I(gcd(a,b)) = I(a) ∩ I(b)

3.6 modular arithmetic

if result / intermediates always taken mod b
for b = 3, then 5 + 8 mod 3 = 1
reduce by b "as we go along"

inverse
a (multiplicative) inverse b iff b * a = 1
a modular inverse iff b * a = 1 mod N
iff gcd(a, p) = 1 then modular inverse exist
iff p prime, then every a < p has modular inverse

units
generate all elements of the group (order = group size)
order of each element has to divide group order

congruence $=_n$
a $=_n$ b iff a-b = k*n for some k
a $=_n$ b iff a mod N = b mod N

Z_n
for partition introduced by $=_n$
$|Z_n|$ = n
(Z_n, +, *) commutative ring (add, multiply)
iff n prime, (Z_n, +, *) is field (ring + inverse)

fermat's little theorem
for p prime, $a^{p-1} =_p 1$ for any $0 < a \in Z_p$
hence identity follows with $a^p =_p$ a
hence inverse(a) follows with a^{p-2}

totient function
totients(n) are all k < n for k not dividing n
ϕ(N) (totient function) is |totients(n)|
for p prime, $\phi(p^k)$ = (p-1)p^{k-1}
hence ϕ(p) = $(p-1) * p^0$ = p-1
for a ⊥ b, ϕ(a * b) = ϕ(a) * ϕ(b)
hence ϕ(p * q) = (p-1)*(q-1)

euler's theorem
for any totative a of n, a{ˆϕ(n)} $=_n$ 1
useful to simplify powers (a{ˆϕ(n)*b + c} = a^c)
or to calculate inverses a^1 = a{ˆϕ(n)-1}

4 RSA

4.1 textbook RSA

N typically 2048bits (hence very large)

construction
KGen chooses random primes p, q of some bitsize k/2
let N = p * q, ϕ(N) = (p-1)*(q-1)
choose d, e such that d*e = 1 mod ϕ(N)
output (sk = d, pk = (e, N))
Enc(pk, m ∈ [1, N-1]) outputs c = m^e mod N
Dec(sk, c) outputs m = c^d mod N

choosing d,e
select e, then use EEA(e, ϕ(N)) to get d
iff e coprime ϕ(N), get e*s + ϕ(N)*t = 1
resolve to e*s = 1 mod ϕ(N)
hence d = s inverse of e

often e = 2^{16+1} chosen
e is prime, likely co-prime to (p-1)(q-1)
encryption becomes fast (because its small)

correctness by eulers theorem
given are $|Z_n|$ = ϕ(N) and d*e = 1 mod ϕ(N)
m^{de} = m{ˆ$1 + k * \phi$(N)} = m * (m){ˆϕ(N)*k}
= m * 1
not applicable to m % p = 0 or m % q = 0

correctness by fermats little theorem
assume m coprime N (hence coprime to p-1, q-1)
d*e = 1 + k * (p-1) * (q-1) for some k
m + m{ˆk*(p-1)(q-1)} mod p = m mod p (same holds for q)
hence m^{de} mod N = m

difficulties
generating random primes of given bitsize
choosing d, e (can pick e randomly, use EEA)
messages need to be encoded in interval [1, N-1]
enc is not randomized (hence not INC-CPA secure)
choosing small d's is insecure

hardness
it should be hard to get d; given e, c and N
solvable by factoring N (which is assumed hard)
solvable by other means so far unknown to be faster

4.2 challenges generating RSA keys

need good source of randomness
need efficient primality test (like probabilistic with low error rate)

repeated usage of same prime
given $N_1 = p_1 * q_1$, $N_2 = p_1 * q_2$, can recover $p_1 = \gcd(N_1, N_2)$
for M distinct N, compute pairwise in O(M^2) or O(M log M) (bernstein)
in 2012, broke 0.5% public keys as randomness generation insufficient
(2^{990} primes of length 128 bits, hence bad randomness most likely)

ROCA attack
p, q generated on low-performance device
but manufacturer of smartcards overoptimized
could recover p, q in some cases

primality tests
require random bases, but some primality tests implemented improperly
like miller-rabin used fixed bases, or others used weak PRGs for bases
hence could construct non-primes that pass the tests

4.3 keysize requirements

if factoring N is easy RSA broken
no iff; might be other required assumptions (but unknown)

integer factorization problem (IFP)
studies for many years, intensively since 1970
best found algorithm so far is number field sieve (1990)
quatum shorr algorithm runs in polynomial time

number field sieve (NFS)
sub-exponential (harder than polynomial, easier as exponential)
$\exp[(c + o(1))(\ln N)^{1/3} (\ln \ln N)^{2/3}]$ for $c = (64/9)^{1/3}$

concrete requirements
512-bit in 2015 for USD 75 on amazon EC2
768-bit 2009-2019 for 2000 core years
795-bit 2019 for 900 core years
829-bit 2020 for 2700 core years
conjecture 1024-bit requires around 2^{80} operations
(at lot, but in reach for NSA with 100 mia budget)
for 128-bits, require 2048 - 3072 bits
see https://www.keylength.com

4.4 problems

malleability
for c = m^e mod N
can choose s, then multiply to c
gives valid ciphertext $(s * m)^e$ mod N
hence attacker can modify plaintext in controlled fashion

small e
for e = 3, and m < $N^{0.33}$
then c = m^3 is over integers (no modular reduction)
small d also insecure, up to d < $N^{0.25}$ (Weiner's attack)
⇒ need message padding

4.5 padding rsa

requirements
introduce randomness into the message
expand short message to full size
destroy algebraic properties between messages (remove malleability)
ultimately want IND-CCA for RSA

4.6 PKCS#1.5

not IND-CCA secure; specification got ahead of research

construction
for k = N/8 (N in bytes), max message size is k-11 bytes
pad(m) = 0x00 || 0x02 || (≥8 random bytes != 0x00) || 0x00 || m

destruction
checks for 0x00 || 0x02 start
checks for at least 8 bytes != 0x00
checks if 0x00 follows before the last byte
return m as everything to the right of 0x00 byte

valid padding p from random string
first two bytes have to be 0x00 || 0x02 → p = 2^{-16}
then 8 bytes not 0x00 (likely), then some byte 0x00 (likely for long keys)
hence random string has p = 2^{-16} of having valid padding

Bleichenbacher attack
requires decryption oracle on input c (realistic assumption)
for some cipher c of message m
attacker asks $s^e * c$ to the decryption oracle
if succeeds, attacker learns that s ∗ m result starts with 0x00 0x02
through adaptive attack, can recover m' in 5-10k queries
vulnerability repeatedly reintroduced in SSL, robotattack.org

4.7 RSA-OAEP

optimal asymmetric encryption padding by bellare and rogaway
IND-CCA under strong assumptions
standardized in PKCS#1 v2.1; not widely deployed
like feistel cipher without keys

construction
for hardness λ_0, λ_1 (adversary cannot perform 2^λ)
need $\lambda - bit$ RSA moduli, CR hash functions G & H
for message m of length n = λ - λ_0 - λ_1
let S_1 = (m || 0^{λ_1}) XOR G(R) (for R random, |R| = λ_0)
let S_2 = R XOR H(S_1)
c = $(S_1||S_2)^e$ mod N
decryption ensures S_1 ends with 0^{λ_1}

rationale
RSA message now randomized, full length
no algebraic properties on messages (bc of the two values, hash functions)
random message decryption will fail with very high p (due to 0^{λ_1})

security
can be proven IND-CCA secure
but need strong G, H assumptions
but need strong number theoretic assumption (stronger than factoring)
if RSA has to be used, best choice as padding

4.8 random oracle model (ROM)

gives strong abstraction of hash functions

construction
given domain → range
assume hash function H is a random function
adversary A must ask oracle to evaluate H
in proofs, results in advantage as can see all queries A makes
can let oracle program responses depending on A's queries

heuristic step
real hash functions such as SHA-256 are fixed
hence (unsound) heuristic step required when applying in practice

controversy
many arguments for and against ROM in security proofs
can make trivially insecure schemes which are provable under ROM
but also enables proofs of in practice secure schemes otherwise unprovable

4.9 RSA-KEM

IND-CCA secure in ROM under RSA inversion assumption
idea is to hash plain so malleability goes away

construction
KGen generates p,q of bitsize k/2
choose d∗e = 1 mod ϕ(N) for N = p ∗ q
let H: {0, ..., N-1} → $\{0, 1\}^k$, sk = d, pk = (e, N)
Encap(pk) chooses z ← {0, ..., N-1}, returns (c = z^e mod N, K = H(z))
Decap(c, sk) computes z = c^d mod N, returns K = H(z)

RSA inversion assumption
for sk = d, pk = (e, N), x ←$ {0, ..., N-1}, y = x^e mod N
A is given (N, e, y)
has to output x (= calculate $y^{1/e}$ mod N)
A wins at least by factoring N (simpler solutions might exist)

RSA inversion hard ⇒ IND-CCA security
assume attacker A breaking IND-CCA of RSA-KEM in ROM
C sends (N, e, y = x^e) to reduction B, expects x back
B maintains (initially empty) list of triplets (c, z, K)
on dec(c) query, check if c exists in triplets
if yes, respond with K
else, choose random K to respond, remember (c, ?, K)
on H(z) query, check if z exists in triplets
if yes, respond with K
else, check if c = z^e mod N exists in (c, ?, K)
→ if yes, update entry to (c, z, K), return K
→ else, select random K, remember (c = z^e mod N, z, K), return K
B checks if at the end (y, z, K) in table, sends z to challenger

B does not care what A outputs

5 discrete logarithms

research first focused on RSA setting
only later discovered than DH can be used similarly

5.1 setting

for p, q large primes for q divides p-1
let k = (p-1) / q

generator g
g = h^k mod p for random h ←$ {0,...,p-1}
iff g != 1, then g builds G_q = {g, g^1, ..., g^q}
(1) all values in G_q are distinct
(2) g^q mod p = 1
(3) ∀ j,k ∈ G_q . j∗k ∈ G_q
hence G_q is cyclic group under multiplication mod p
with g its generator and size $|G_q|$ = q

example
p = 29, q = 7 for 7 divides 28
k = 28 / 7 = 4
g = 16 = 2^4 for random h = 2
G_q = {16, 24, 7, 25, 23, 20, 1}
verify that 16^7 = 1, 24∗7 mod 29 = 23

check group membership X
required in some protocols to prevent small subgroups attacks
ensure that X^q mod p = 1

5.2 hardness

discrete logarithm problem (DLP)
let (p, q, g) be as introduced
let y = g^x mod p for uniform random x
find x
(like finding the logarithm to the base g)

computational diffie hellman problem (CDH)
given (p,q,g) and x = g^a mod p, y = g^b mod p
find z = g^{ab} mod p
CDHP > DLP (as DLP(y) → b, then x^b = z)
DLP > CDHP is not proven in general, but widely believed
used in diffie-hellman key exchange

decisional diffie-hellman problem (DDH)
given (p,q,g) and unf. random a, b, c
distinguish (g^a, g^b, g^{ab}) from (g^a, g^b, g^c)
used in ElGamal public key encryption scheme

5.3 solving DLP

intense analysis from math / computer science over last 40 years
solve p with FFS, q with $polland-\rho$
for 80bits, need p > 1024 bits, q > 160bits (most real-world deployments)
for 128bits, need p > 3072 bits, q > 256bits
quantum algorithm Shor breaks DL (as well as RSA)

functional field sieve (FFS)
sub-exponential (harder than polynomial, easier as exponential)
exp[(1+o(1)).c (ln p)$^{1/3}$ (ln ln p)$^{2/3}$] for c = (32/9)$^{1/3}$
runtime similar to number field sieve, but different constant

$polland-\rho$
exponential in log p (hence doubling bit size, doubling runtime)
O($q^{0.5}$)

5.4 diffie hellman key exchange (1976)

public key method to agree on shared secret
released in 1976 by diffie / hellman, launching public key crypto
relays on hardness of CDHP
original paper describes public key lookup out of directory

construction
let (p, q, g) be as introduced
each user U_i picks x_i ←$ {0, ..., q-1}
calculates public key $Y_i = g^{x_i}$ mod p
put public key into for directory
users U_i, U_j can calculate shared key K = ${Y_j}^{x_i} = {Y_i}^{x_j}$
use K as seed for key derivation function (KDF)

modern view with exchange (and MitM)
users agree on (p,q,g), generate fresh x_i and exchange $Y_i = g^{x_i}$
Y_i, x_i regarded as ephemeral (hence used only once)
but active attacker can MitM during exchange
need authenticity of Y_i and Y_j (by MAC or digital signatures)

authenticate with MAC
could authenticate public values Y_i, Y_j with MAC
requires shared MAC key
still beneficial as enables forward secrecy
e.g. even if MAC key later compromised, shared DH key still secure

authenticate with digital signature
could authenticate public values Y_i, Y_j with signature
requires detection of authentic signatures
typically done using certificates, CAs & PKIs

5.5 ElGamal (1985)

essentially a one-time DH key exchange
requires m to be encoded in G_q
relays on hardness of DDH
IND-CPA, but not IND-CCA (use only as KEM)

construction
let (p, q, g) be as introduced
KGen chooses x ←$ from {0, ..., q-1}
outputs (pk = (X = g^x), sk = x)
Enc(X, M ∈ G_q) chooses r ←$ from {0, ..., q-1}
outputs (Y = g^r, M ∗ (Z = X^r)) (blinds M with Z)
Dec(x, C = (Y, C')) ensures Y ∈ G_q (else terminates)
output M = C' ∗ $(Z' = Y^x)^{-1}$

IND-CPA under CDH
cyphertext includes M ∗ Z for Z = g^{xr} mod p
can replace g^{xr} by g^c for c random by CDH
as g^c uniformly random, M ∗ g^c is too
hence M perfectly hidden

IND-CCA adversary
enc(m_0, m_1) → (Y, C) for m_0 != m_1
dec(Y, C ∗ g^2) → m'
checks if m' / g^2 == m_0 then b = 0 else b = 1

5.6 diffie hellman integrated encryption scheme (DHIES)

IND-CCA in the random oracle model
any M (not necessarily ∈ G_q)
requires IND-CPA encryption and SUF-CMA MAC (AE)

construction
let (p, q, g) be as introduced, H hash function
KGen chooses x ←$ from {0, ..., q-1}
outputs (pk = (X = g^x), sk = x)
Enc(X, M as bitstring) chooses r ←$ from {0, ..., q-1}
set H((Z = X^r), X, (Y = g^r)) = k
split K into encryption key K_e and MAC key K_m
output (Y, C' = SymEnc(K_e, K_m, M))
Dec(x, C = (Y, C')) ensures Y ∈ G_q (else terminates)
(K_e, K_m) = H((Z = Y^x, X, Y)
return M = SymDec(K_e, K_m, C')

KEM/DEM instance
value K is encapsulated key (KEM)
can use any AE scheme for SymEnc/SymDec

6 digital signatures

guarantee integrity of message m

6.1 application of signatures

suites by NIST, NSA, NESSIE
recommend are ECDSA, RSA-PSS, PKCS#1 v1.5 with RSA

use-cases
public verification of message authenticity / integrity
code-signing
entity authentication (sign challenge to prove key possession)
certification systems (signatures to authenticate other keys)

cryptography signatures
some legal frameworks in place in switzerland & EU
identification cards deployed in belgium, estonia, ...
requires physical security, tamperproof hardware, special terminals
human understanding/usability major barrier

6.2 definition

KGen \$→ (sk, vk)
Sign(sk, m) → σ for m ∈ $\{0,1\}^*$
Vfy(vk, m, σ) → 0|1

correctnes
for all (sk, vk) of KGen
if σ = Sign(sk, m), then Vfy(vk, m, σ) = 1

non-repudiation
if vk bound to identity & EUF-CMA signature scheme
then user cannot deny having created signature
MAC cannot offer this as many parties have shared key

legal
difficult to enforce non-repudiation
as requires proving (sole!) ownership of private key

6.3 security notions

assume that receiver has authentic verification key vk
it must be hard without sk to find m∗ and σ∗
such that Vfy(sk, m∗, σ∗) outputs 1

single-user security definition
only says something about security of specific sk / vk
but might be able to get valid sk/vk pair
then forge signatures under differnet vk∗

unforgeability chosen-message attack (UF-CMA) game
challenger creates (sk, vk) ←\$ KGen
challenger provides vk to adversary
adversary can use signing oracle sign(m) = Sign(sk, m)
(no verification oracle like MACs as verification public)
adversary wins if outputs (m∗, σ∗)
for m∗ not in queried values, Vfy outputs 1

(q_s, t, ϵ)-UF-CMA
adversary querying q_s, running in time t
for m∗ fresh message (never queries)
$\text{Adv}_{SIG}{}^{UF-CMA}$ = Pr[Vfy(vk, m∗, σ∗) = 1] = 1] < ϵ
UF same as EUF (existential universal unforgeable)

strong UF-CMA (SUF-CMA)
adversary wins if (m∗, σ∗) different
any UF-CMA adversary breaks SUF-CMA (hence SUF-CMA ⇒ UF-CMA)

EUF-CMA ⇒ SUF-CMA
SUF is stronger (easier to break for adversary)
any EUF adversary breaks SUF
equivalent for unique signature schemes

6.4 digital signature algorithm (DSA)

introduced by NIST in 1991
subsequently updated for different keysizes, hashes
cannot easily be turned into encryption scheme (export restrictions)
requires CR-hash H and DLP hardness

setup
160-bit prime q, 1024bit p such that q | p-1
p,q,g shared by users; around 80bits of security
KGen selects random 1 ≤ x ≤ q-1 (without 0)
output (sk=x, $vk = g^x$ mod p)
Sign(m,sk) generates random 1 ≤ k ≤ q-1
let r = (g^k mod p) mod q
let s = k^{-1} ∗ (H(m) + x∗r) mod q
output σ = (r,s)
Verify(pk, m, σ = (r,s))
ensure 1 ≤ r, s ≤ q-1 (must check!)
let w = s^{-1} mod q
let u_1 = w∗H(m) mod q, u_2 = w∗e mod q
accept if (g^{u_1} ∗ y^{u_2} mod p) mod q = r

correctness
$g^{u_1} * y^{u_2}$ = g{ˆw∗H(m)} ∗ g{ˆx ∗ wr} = g^k mod p = r
because $g^{w(H(m)+xr)} = g^k$

signature size
2 ∗ 160 bits at 80bits security
notably much smaller than RSA signatures
signing requires only exponating a short exponent k (160 bits)

security
relays on linear equation s with two unknows (r, x)
but formal & clean security proof known
generic attacks (solve DLP, O($q^{0.5}$) bruteforce, hash collision)

randomness failure
suppose same k / x used with two different messages
producing valid signatures $\sigma_{1(r_1,s_1)}$, $\sigma_{2(r_2,s_2)}$
can detect when $r_1 = r_2$ (as k equal by assumption)
with s_1 - s_2 = $k^{-1(H(m_1)-H(m_2))}$ can recover k (as m_1, m_2 known)
known k allows to extract x from s_1
OpenSSL bug (2008), PlayStation 3 (2010), Android (2013)

related randomness problems
only need to predict few bits to attack possible (5 MSB enough)
like timing attack measures fast signature if MSB are 0
weak randomness generator, relation between bits same problem

hedging DSA against randomness failures
generate k using pseudo-random function (requires secret key k)
k = $F_{K(vk||m)}$ (note that same message ⇒ same k ⇒ same σ)
general way to solve randomness problem

6.5 textbook RSA signatures

construction
KGen chooses p & q, sets N = p∗q
choose ed = 1 mod ϕ(N)
output (vk = (N,e), sk = d)
Sign(sk, m) outputs $\sigma = m^d$
Vfy(pk, m, σ) checks σ^e = m
correctness like in RSA

problem
forgery of new signature trivial
multiply σ with some other value

6.6 RSA full-domain hash (RSA-FDH)

requires CR-hash function H
H destroys multiplicative structure, allows signing long messages
UF-CMA secure if RSA inversion hard, H random oracle
weak proof (reduction not tight)

construction
KGen chooses p & q, sets N = p∗q
choose ed = 1 mod ϕ(N)
output (vk = (N,e), sk = d)
Sign(sk, m) outputs $\sigma = H(m)^d$
Vfy(pk, m, σ) checks σ^e = H(m)

security
s signing queries, h hash queries
$\text{Adv}_{RSA-FDH}{}^{UF-CMA}$(A) ≤ ($q_s$ + q_h) $\text{Adv}_{RSA-INV}$(b) - 1/N
q_h is potentially high (offline hash computation)
tighter proof replaces $q_s + q_h$ by q_s (2000)

bound calculation
for $Adv_{RSA-FDH}{}^{UF-CMA(A)} \leq (q_s + q_h) * Adv_{RSA-INV(B)}$
realistic to generate $q_h = 2^{80}$ hashes
then Adv(A) ≤ 2^{80} ∗ Adv(B) <= 2^{-48} => too low
but better reduction available with $q_h = 0$
then we limit signing queries to 2^{32} (realistic, as these are online)
results in Adv(A) ≤ 2^{32} ∗ Adv(B) <= 2^{-96} => good enough!

proof scetch
C gives (N, e, y), expects x of x^d = y (e-th root)
B chooses j (prediction which message A will forge)
B passes (N, e) to A
A queries sign(m) and hash(m)
on hash(m_i), iff i=j then x else y_{i^e} for y random
on sign(m_i), iff i=j then B must abort, else y_i
B does for every sign query a hash query (consistency)
A outputs (σ∗, m∗)
forgery successful if A forged j'th hash query (1/(q_{h+q_s}))
A might also predict H(m∗) output (1/N)
$\text{Adv}_{RSA-FDH}{}^{UF-CMA}$(A) ≤ ($q_s$ + q_h) $\text{Adv}_{RSA-INV}$(B) - 1/N

6.7 hash-based RSA signatures

in use and widely standardized
but no security proof

signature construction
use deterministic padding scheme pad and hash function H
like $\sigma = pad(H(m))^d$ mod N
adapt Vfy correspondingly

PKCS#1 v1.5 padding
00 01 FF .. FF 00 || c || H(m) for constant c
security proof unknown, no known attacks
padding check/removal often implementation issues
forgery possible if constant part of padding too short

6.8 RSA-PSS

for RSA signatures, RSA-PSS the right choice
tight security reduction

signature construction
for H, G_1, G_2 hash functions
s = H(m || r) for some random r
t = $G_{1(s)}$ XOR r, u = $G_{2(s)}$
σ = (0 || s || t || u)
adapt Vfy correspondingly

security
assuming G_1, G_2, H behave like random functions
UF-CMA can be tighlty related to RSA inversion
can instantiate with "ordinary" SHA-256 (no full domain hash required)

6.9 advanced signature variants

blind signatures
A lets blinded message signed by B (B does not learn message)
used in anonymous credential systems

group signatures
anyone from group of uses can sign
signer might be revealable by some group manager

threshold signatures
any k out of n parties can sign
k-1 or fewer cannot

others
proxy (delegate limited signature capability to others)
ring signatures, multi-signatures, aggregate signatures
standardized in PKCS#1 v2.1

7 elliptic curve cryptography

can define DL-based algorithm over any cyclic groups
elliptic curve is candidate with no known sub-exponential algorithms
only generic DL-breaking algo known, runtime is O($n^{0.5}$)
allows to use smaller bit-sizes, improving performance
proposed 1985, usage started around 2015

7.1 shorter key length

key size comparison (keylength.com)
required security level compared to sizes by scheme
RSA modulos | DL field / subgroup size | elliptic courve
80 bits → 1024 | 1024, 160 | 160
112 bits → 2048 | 2048, 224 | 224
128 bits → 3072 | 3072, 256 | 256
256 bits → 15360 | 15360, 512 | 512
cost of exponentiation in RSA/DL rises qubicly
note that EC is optimal, linear rise

RSA / DL
besides generics (Baby-steps-Giant-steps, $pollard - \lambda$, $pollard - \rho$, ...)
more efficient algorithms known (number-field sieve and variants)
hence need to choose large modulos / DL field

7.2 definition

for some field F, curve is set of (x,y) ∈ F × F

weierstrass form
E = {(x,y) ∈ F × F | $y^2 = x^3$ + ax + b ∪ {O}}
point O is "point at infinity", no coordinate representation
non-triviality requirement $4 * a^3 + 27b^2$!= 0
note that due to y^2, get symmetry above and below x axis

other forms
montgomery form allows to calculate only on x
edwards form avoids side-channel attacks

7.3 math

example E
$y^2 = x^3$ + 2x + 4 mod 5
evaluate x = {0, 1, 2, 3, 4} to get y^2 = {4, 2, 1, 2, 1}
from possible y^2 values ({4,2,1}) find roots (where root possible)
4 → (2^2 mod 5, 3^2 mod 5), 2 → no root, 1 → (1^2 mod 5, 4^2 mod 5)
get points (x,y) → (0,2),(0,3),(2,1),(2,4),(4,1),(4,4)
with O, get 7 points on E

addition law
additive identity is O
additive inverse of P is -P = (x, -y); O = -O; P + -P = O
calculate P + Q = R using geometric construction
drawing line through P, Q (or tangent, iff P=Q)
intersect with curve, then mirror ("inverse") at x axis
resulting point is R

group
addition law turns elliptic curve field into a group
group operation is +, "adding" points
for generator P, group generated by P, P+P, P+P+P, ...
group order is number of points on the curve

scalar multiplication
[k]P for adding P to itself k times
like [2]P = P+P
note that [k]P != (k∗x, k∗y)

double-and-add
like square-multiply of multiplicative setting
for 5 = 101
1: O → [2]O + P = P
0: P → [2]P
1: [2]P → [4]P + P = [5]P
⇒ must not leak addition count by side-channels

discrete logarithm problem (ECDLP)
let E elliptic curve
let P point of prime order q
let Q = [x]P where x is uniform random value {0,1,..,q-1}
given E, P, Q, find x
only generic DLP algorithms known in O($q^{0.5}$)

7.4 choosing curves

decide field F (usually prime field for some prime p)
decide curve E over F (parameter chosing difficult)
find base point P of large prime order q
implement scalar multiplication arithmetic (but side channels)
much easier to rely on standardized curves by trusted sources

hasse-weil bound
determines number of n points for field of prime order p
p+1 - 2∗sqrt(p) ≤ n ≤ p + 1 + 2∗sqrt(p)
for large p, sqrt(p) factor irrelevant

selection considerations
prime order curve (n prime) to maximize against generic algorithms
otherwise ensure "co-factor" h small in n = h∗q
use Schoof-Elkies-Adkin (SEA) algorithm to compute #points efficiently

base point selection
for E defined over F having n points, n having large prime divisor q
choose some random P != O by picking x, then solving for y
succeeds with p = 0.5, as half of non-zero elements mod p are square

point compression
naively would store (x, y), requiring 2 log_2 (p) bits
but enough to store x, then use equation to recover y
add 1-bit ("sign-bit") to differentiate between y and p-y

key-pair generation
for E over F with n points
let q be prime divisor of n, P points of order q
choose random scalar k ∈ {0, ..., q-1}
set Q = [k]P
output (sk = k, pk = Q)
hardness of getting sk out of pk based on ECDLP assumption

NIST P-256
in field with p ˜ 2^{256}
$y^2 = x^3$ + ax + b with a = -3, b some truly large number
base point with prime order q, h = 1
both p, q have 256 bits
but chosing of a and b not properly motivated (backdoor?)
used in TLS 1.3

berstein curve 25519
p = 2^{255} - 19 (bc closest prime to 2^{255})

y = x^3 + 48662x^2 + x (bc 48662 smallest number with target performance/security)
montgomery form (fast modular reduction, only scalar multiplications)
but group order is not prime (has co-factor of 8)
bit less than 128 bits security, faster than P-256
used in TLS 1.3

7.5 cryptography

can translate DLP setting schemes into ECDLP
like DHE ⇒ ECDHE, DHIES → ECIES and DSA → ECDSA

elliptic curve diffie-hellman ephemeral (ECDHE)
given curve E, base-point P of prime order q
A choose random x, B chooses random y
A sends [x]P to B, B sends [y]P to A
both can calculate [x][y]P, resp [y][x]P
security by decisional diffie hellman

ECIES
let (P, F, q) be as introduced, H hash function
KGen chooses x ←$ from {0, ..., q-1}
outputs (pk = (X = [x]P), sk = x)
Enc(X, M as bitstring) chooses r ←$ from {0, ..., q-1}
set H((Z = [r]X), X, (Y = [r]P)) = k
split K into encryption key K_e and MAC key K_m
output (Y, C' = SymEnc(K_e, K_m, M))
Dec(x, C = (Y, C')) ensures Y on curve with order q (else terminates)
(K_e, K_m) = H((Z = [x]Y, X, Y)
return M = SymDec(K_e, K_m, C')

ECIES performance
longer ciphertext (elliptic curve point +256bits, MAC tag +128bit)
encryption requires 2 scalar multiplications
decryption requires 1 scalar multiplications

ECDSA
signature is (r,s) for r,s integers mod q (512 bits)
requires per-signature nonce, else fatal loss
if some bits known, can recover key like in DSA
malleability (for valid (r,s) → (r, -s), hence SUF broken)
UF-CMA-security proven in generic group model

7.6 slow take-up of ECC

discovered by koblitz, miller (1980)
widespread only in 2010 (30 years!)
20% 2013 (snowden leak), 70% 2016, 90% of 2018
in TLS 3.0, no RSA anymore

slow adoption reasons
mathematical, implementation complexity (relative to RSA)
security uncertainty due to marketing feud (RSA vs Certicom)
lack of mature standards (developed in 2000s)
patent situation (Certicom threatening to sue others)
hard to displace exiting technology

drivers of adoption
improved performance over RSA
ECDHE provides forward security (vs RSA), usecase for TLS
patent situation clarified due to deal with US gov, expiration
mass-scale adoption in crypto currencies

8 key management

secure administration of cryptographic keys
cryptography shifts problem "securing data" ⇒ "managing keys"

8.1 key management system (KMS)

any system managing keys throughout their live
counters threads like compromise / unauthorized used of keys

requirements
symmetric keys secret
public keys authentic, private keys secret
assurance of purpose (encryption, MAC, ...?)

means
technical (special hardware devices)
process (dealing with lost keys, ...)
environmental (controls depending on physical location)
human factors

aspects
generation
distribution and initialisation
usage and scope
storage, backup and recovery
replacement, revocation and destruction

key lifecycle
pre-operational phase (key not yet available)
operational (key used for intended usage)
post-operational (key used for access to protected records)
destroyed (key deleted, encrypted records inaccessible)

8.2 key generation

out of randomness generation
memory allocator that outputs random location (but insecure)
intel RD RAND (but specification unpublished)
extract entropy out of images of lava lamps
quantum RNG (hardware measuring light stuff)
might want multiple parties contributing

derive out of PIN / password
resulting keys only as strong as starting values
but often only source material of dubious cryptographic strength
use salting, iteration to slow down dictionary / bruteforcing attacks
like PAKE (password-based authentication key exchange)

8.3 key out of master secret

assume master key already distributed

key derivation function (KDF)
K = KDF(master key, info) for info containing context
instantiate using hash function or encryption
but need pseudo-randomness assumption on output of used primitive
provides forward security of previous keys
requires synchronization with receiver

EMV (bank cards)
bank has few master keys, derives K for each user
user gets card with embedded K
used to compute MAC values in transactions

TLS 1.2 (RFC 5246)
keyblock = PRF(master secret, "key expansion", server + client random)
need to iterate over PRF until enough bits produced
then extract for each party IV, MAC key & encryption key
depending on cipher suite, iteration / splitting different
could have enabled attacks, but never abused (fixed in TLS v1.3)

TLS 1.2 PRF
PRF(secret, label, seed) = P_{hash}(secret, label + seed)
A(0 = seed), A(i) = $HMAC_{hash(secret, A(i-1))}$, ...
P_{hash}(secret, label) = $HMAC_{hash}$(secret, A(1) + seed)
HMAC uses hash function given by cipher suite
assumes HMAC being PRF (provable in ideal cipher model)

HKDF ("Extract-then-Expand" KDF)
requires input key material (IKM)
iff IKM not high-entropy preprocess with PRK = $HMAC_{hash}$(salt, IKM)
T(0) = empty string (base case)
T(1) = $HMAC_{hash(PRK, T(0)|info|0x01)}$ (for context field info)
T(2) = $HMAC_{hash(PRK, T(1)|info|0x02)}$ (and so on, arbitrary length)
assumes HMAC randomness extractor (due to PRK step; statistical)
assumes HMAC PRF (for T(i) to be useful; computational)

8.4 asymmetric key generation

requires large primes

construction
generate random odd number, setting most & least significant bit
(to get correct length, and uneven number)
try with some small divisions as first filter
then use probabilistic miller-rabin

miller-rabin primality test
requires random base as input to work properly
then for 10^{24} bit prime, 3 iterations, faillure p < $1/2^{128}$
on adversarial inputs, faillure p < 1/4

cost
prime generation is expensive
$2^{1014.5}$ different 1024-bit primes
hence two uses generating same prime improbable
but requires around 350 trials until prime numer is found

while single test cheap, scales poorly

8.5 distributing keys

low-tech examples
send courier with key material (like RU / USA 1963 - 1980)
or send over postal mail (like E-Voting)

three-layer master key / session key scheme
KKM (master) to encrypt KK or KDs (manual exchange)
KK to encrypt KD values (automatic exchange)
KD as working key (changed very often)
but inefficient in large, many-to-many systems

hybrid public/symmetric key scheme
use public key to encrypt symmetric key (primitive is a KEM)
requires only authentic public keys to be distributed
authenticity provided by certificate authority (CA)

unique key per transaction
derive new key for each usage
like KDF taking master key & transaction counter
useful for insecure environments, side channel attacks harder

diffie hellman key exchange
public key method to agree on shared key
but MitM attack if not properly authenticated

quantum key distribution
use quantum physics principles to distribute keys
requires authentic channel, but then delivers unconditional security
steady development (since 1988 distance/throughput gradually increased)
china, EU invest heavily in research
but range-limited (few-hundred km over optic fibre, longer in vaccuum)
but not end-to-end secure (as need repeaters for long distances)
but low bit-rates (function of distance, not sufficient for one-time-pad)
but requires authentic channel (which again requires pre-agreed key)
but expensive devices with side-channel risks
unclear real-world value, requires combination with conventional primitives

8.6 key storage

tamper-resistant hardware security module (TRSM, HSM)
smart card / personal token
outside TRSM but encrypted and/or split into components
in practice often stored in memory, protected only by OS

hardware security modules (HSM)
usually store local master key used in processing
security through restricted function range
physically secured as specified in FIPS 140-2 (tamper-resistance)
high-value, very expensive devices

tokens
hardware (PC cards, smart cards, USB sticks, ...)
software (PKCS#12, proprietary methods)
boom due to crypto currency bubble

hidden in software
cheap, requires only some obfuscation
but dangerous (reverse engineering)
might encrypt keys, but then require decryption keys

8.7 key usage

principle of separation
cryptographic keys should only be used for intended purpose
requires defined & limited purposes
derive different keys using the info field of KDF

reason for separation principle
primitives might interact unexpected (like CBC-enc and CBC-mac)
encryption/authentication may have different lifecycles
less damage from key compromise
but increases key management effort

controlling key usage
derive labeling scheme and bind labels to keys
enforce that only keys with proper labeling are used
might label ownership, validity, indended use & algorithm

8.8 key change

planned (limited lifetime, data limit for specific key)
unplanned ((potential) compromise, departure of employee)

impact
minimally have to generate & establish new key
cost of new hardware, migration, trust, reputation

destruction
when key expires / is revoked
might need overwrite memory or physical hardware destruction

policies, practices, procedures
policy describes overall strategy at organisation level
practices describe tactics to archieve policy
procedures with step-by-step tasks to implement practices

standards
NIPS SP 800-57
any many, many more

9 entity authentication

assurance about identity of partner at some point in time
data origin with recency also results in entity authentication
respective to role (A ⇒ B does not necessarily imply B ⇒ A)

9.1 MAC scheme

requires unforgeable MAC and unpredictable random R

construction (server ⇒ client)
client & server share mac key K_X bound to client X
client requests authentication
server sends challenge R $\leftarrow \{0,1\}^{128}$
client responds with $\tau \leftarrow$ Tag(K_X, R)
server accepts if Vfy(K_X, R, τ)

predict challenge
can MitM client <=> attacker <=> server
when client requests authentication, predict future R to sent to client
later attacker requests authentication at server
can use client answer from first run to answer server challenge
(note that time-shift required to break security, live MitM not enough)

two-way construction (server <=> client)
client & server authenticate each other with same key
reflection attack by asking challenge of server to itself
prevent by using different keys (key separation principle)
prevent by including intended recipient partners in MAC

timestamps (server ⇒ client)
client & server share mac key K_X bound to client X
client sends $\tau \leftarrow$ Tag(K_X, t) for timestamp t
server checks if t recent, Vfy(K_X, t, τ)
log received messages to prevent (recent) adversary reply

9.2 signature scheme

requires unforgeable signature and unpredictable random R

construction (server ⇒ client)
client has keypair (sk_X, vk_X), identity X & certificate of identity
client requests authentication & sends certificate
server validates certificate (chain) & sends challenge R $\leftarrow \{0,1\}^{128}$
client responds with $\tau \leftarrow$ Sign(sk_X, R)
server accepts if Vfy(vk_X, R, τ)

9.3 gsm entity authentication

SIM card has IMSI, key K (128bits)
network provider has mapping of IMSI → key K

construction
phone sends IMSI to visited network
visited network forwards to home network (network provider)
generates random challenge Rand, XRES = RPF(K, Rand)
SIM is forwarded Rand, and replies with RES
iff RES == XRES, then authenticated

reality adaptations
IMES is sent anonymized
home network generates multiple Rand, XRES pairs (to avoid roundtrips)
network also authenticated (to identify fake base stations)

key establishment construction
home network additionally generates K_c = PRF2(K, Rand)
K_c also forwarded to visited network (but SIM card deduces self)
K_c initializes stream cipher for wireless encryption

analysis
database of IMSI → key K is single root of failure
visited network has to be trusted for encrypted exchange
wireless portion is encrypted, nothing else

9.4 authenticated key exchange (AKE)

distribute key material against dolev-yao adversary
party(ies) get assurance with whom key established

PKE construction
server has (pk, sk), identity Y
client requests authentication
server responds with Cert(Y, pk_Y, appropriate chain)
client verifies chain, choose random K
responds with C ← Enc(pk_Y, K)
only server can decrypt
⇒ but server not authenticated to client yet

PKE construction (server ⇒ client)
server has (pk, sk), identity Y
client requests authentication
server responds with Cert(Y, pk_Y, appropriate chain)
client verifies chain, choose random K, random R $\leftarrow \{0,1\}^{128}$
responds with C ← Enc(pk_Y, K), R
server decrypts K, K_a ← KDF(K, "auth"), τ ← PRF(K_a, R)
client receives τ, checks for validity

TLS 1.2
used scheme similar to PKE construction w/ server authentication
but no forward security (as can deduce previous session keys)

EC construction (server ⇒ client)
client requests authentication with random R $\leftarrow \{0,1\}^{128}$
server selects curve parameters (likely standardized curve)
chooses secret y ∈ {0, ..., q-1} to get Q = [y]P
generates signature τ over all curve parameters, Q, R
client receives all (except y)
verifies curve parameters (Q on E, Q order q, valid standard curve)
chooses own secret x to get S = [x]P
generates session key with HKDF(K_{raw} = [x]Q, "session key")
client sends S to server
server validates (S on E, S order q)
provides forward secrecy

EC construction with long-term keys (server <=> client)
client has (a, A = [a]P), server (b, B = [b]P)
client chooses x, sends X = [x]P & certificate of identity
server validates certificate, X on E
chooses y, sends Y = [y]P & certificate of identity
client validates certificate, Y on E
both agree on K_{raw} = [a]B || [x]Y, used for KDF(K_{raw}, "sk")

key compromise impersonation (KCI)
when key compromised, other parties can be impersonated to self
(additionally of course to impersonating self to others)
like last EC construction

10 SSL/TLS

communicate over internet ("secure channel between two peers")
preventing eavesdropping, tampering, message forgery
between application (HTTPS) and TCP layer
billions of devices, various implementations
certification authorities are single point of failure

10.1 history

1994-1996 SSL 1.0-3.0 (all considered broken)
1999 TLS 1.0 RFC 2246 by IETF (like SSL 3.1)
2006, 2008 TLS 1.1, TLS 1.2 (small improvements)
2018 TLS 1.3 (big rework)

10.2 high-level goals

secure against attacker with complete control of network (Dolev-Yao)
only requires in-order, reliable data stream

authentication
server side of channel is always authenticated
client authentication optional
using asymmetric crypto (signatures), symmetric pre-shared key

confidentiality
data only visible to endpoints
length of data not hidden (but padded)

integrity
data sent cannot be modified without detection

10.3 main componenets

handshake protocol
negotiates security parameters
authenticates peers
establishes key material for daa protection

record protocol
exchanges data confidential and integrity
protects using key material from handshake

10.4 negotiation (v1.2)

client proposes list of ciphers, server picks
then agree on key

TLS-KEX-AUT-WITH-CIP-MAC format
KEX for key exchanges (rsa, dhe, ecdhe)
AUT for authentication (rsa, dss, ecdsa)
KEX & AUT for handshake
CIP for cipher (AES-128-CBC, AES-256-GCM)
MAC for hash function within HMAC (MD5, SHA, SHA256)
CIP & MAC to encrypt records
like TLS-RSA-WITH-AES-128-CBC-SHA

handshake
client sends ClientHello with cipher suites
server responds with ServerHello (specific cipher) & key exchange data
server might include certificate and certificate request
client responds with ClientFinished & key exchange data
client might include certificate
server responds with ServerFinished

example TLS-RSA-WITH-AES-128-CBC-SHA
client sends TLS-RSA-WITH-AES-128-CBC-SHA, random r_c
server responds with TLS-RSA-WITH-AES-128-CBC-SHA, random r_s, SCRT
for SCRT (server certificate) public key with certificate chain
client derives generates preMS (out of state, randomness)
client derives MS = PRF(preMS, r_c || r_s)
CKX = RSA.Encrypt(pk, preMS) under pk of server
client sends CF = PRF(MS, client, H(transcript)), CKX
server responds with PRF(MS, server, H(transcript))
server authenticates by proving decryption of preMS
note that MS used twice (in PRF) for different purpose (double usage)

example TLS-DHE-RSA-WITH-AES-256-GCM-SHA384
after picking cipher suite, define params = courve parameters
server additionally sends g^y and signatures over r_c, r_s, params
server already authenticated due to the signature
preMS by g^{xy} (client again picks secret g^x)
after preMS establishment, same finish as before

10.5 record protocol (v1.2)

payload (stream) divided in segments
prefix length, sequence number & MAC (like HMAC-SHA1 MAC)
encrypt payload, MAC tag, padding (like AES128-CBC)
note insecure MAC-then-encrypt scheme used

10.6 additional features (v1.2)

session resumption (abbreviated handshake, parallel connections)
renegotiation (change cipher within session, like late authentication)
extensions (AEAD, ECC, some security-relevant patches)

10.7 security review (v1.2)

component layers
crypto primitives (RSA, ACDH, HMAC, MD5, ...)
ciphersuite details (data structures, padding, ...)
advanced functionality (negotiation, key reuse, compression, ...)
libraries (OpenSSL, LibreSSL, GnuTLS, ...)
applications (browsers, web servers, SDKs, protocols)
attacks found in each layer, requiring hardening

RSA PKCS not CCA secure
error signal of wrong padding allowed to construct decryption oracle

flaw known at design time, so TLS tried to hide error signal
but improperly hidden, downgrade attacks

MAC-then-Encrypt
padding vs MAC error allows to construct decryption oracle
flaw discovered, TLS tried to hide error signal
but lucky13 attack

downgrade attack (design issue)
EXPORT cipher suites by export restrictions US (requiring weak crypto)
client requests DH suite, attacker adds EXPORT to name
server responds with weak group, attacker removes EXPORT
client will not detect, as cipher suites not part of signature
by logjam 15, "how diffie hellman fails in practice"

buffer over-read (implementation issue)
heartbeat extension (client sends payload, requests pingback)
but client requestes bigger pingback than send
server appends memory dump
by heartbleed 2018

10.8 design goals (v1.3)

clean up
removed broken features (compression, renegotiation)
cleaner key derivation with extranct-then-expand HKDF
removed statis RSA/DH to always get forward secrecy
hardened negotiation to prevent downgrades
remove flawed / unused crypto features
like DES, RC4, ... encryption, only AEAD remains
like MD5, SHA1 hash functions
like kerberos, RSA PKCS key transport

improved latency
first handshake in TLS 1.2 only to learn server capabilities
instead always use ECDHE, send several possible shares, server picks
main handshake only 1-RTT, repeated connections with 0-RTT

improved privacy
full handshake in TLS 1.2 in clear
most of handshake in TLS 1.3 now encrypted

continuity
interopability with previous versions / use cases
indeed much faster adoption rate than TLS 1.2

security assurance
formal analysis of changes
symbolic, computational and pen-and-paper proofs

10.9 design (v1.3)

handshake (single round trip)
client sends r_c (hello) and client key shares (g^x, multiple variants)
server responds r_r (hello), specific key share (g^y)
client & server derive handshake traffic key tk, further messages encrypted
server sends certificate, signature & MAC of whole transcript
client answers with certificate, signature & MAC of whole transcript
and already includes data
hence only single additional roundtrip before data exchanged

secrets agreed upon
handshake traffic key to encrypt parts of handshake
application data traffic key to encrypt traffic
resumption master secret (RMS) to continue sessions
exporter master secret (EMS) for additional key material
EMS used in upper layer application, industry use-case

10.10 provable security (v1.3)

general process
describe abstract protocol
define security
reduce to assumptions

define security
multi-stage key exchange security
consider dolev-yao attacker (eavesdrop / active attacks)
might corrupt parties, reveal some session key
then adversary has to decide real key from random bitstring

model adversary actions
gets protocol actions as rewrite rules
passively observes messages or constructs / replaces messages
might be able to reveal keys of participants
after actions, can challenge unpowned participant
then has to decide whether challenge is random or key

security properties checked
forward security after long-term key reveal
key in-dependence in derivation
varying types of authentication
0-RTT keys (which may support weaker guarantees)

10.11 0-RTT handshake analysis (v1.3)

client uses RMS to encrypt payload in first request
but server must detect fresh request (to avoid repeating executions)

advantage
0-RTT for authentication, key is already authenticated
no public key crypto used anymore

suggested mechanism
single-use tickets (allow RMS to be used only once)
recording (reject by unique identifier)
freshness checks (reject based on time)
in practice, libraries choose different solutions

generic state loss attack
attacker is assumed to force state loss (like distributed servers)
then server does not remember log of already received messages
to defend, server must reject 0-RTT after recent state loss
so client/server agree on new key, client resends payload under new key

key schedule
transport key derived using HKDF
key schedule core accumulates secret inputs
key schedule frontend extracts context-separated keys

security properties
random-looking, independent keys
mutual authentication relative to PSK
forward secrecy (not for 0-RTT)
replayable 0-RTT keys
primarily relays on HKDF security, but also on HMAC

10.12 record analysis (v1.3)

record protocol
payload + optional padding
encrypted with AEAD scheme into ciphertext
prepended with TLS v1.2 style prefix for transport layer

security notions
add statefulness (prefix sf) to known notions
IND-sfCPA, IND-sfCCA, INT-sfPTXT, INT-sfCTXT
to get security against chosen ciphertext fragement attacks

key switching within data stream
for forward security, even within streams
for unlimited message encryption length

DTLS
for TLS running over unordered protocol (like UDP)
detect repeated forgeries leading to security degradation

11 signal messaging protocol

two-party asynchronous E2EE message protocol
used by signal, whatsapp, wire, ...
Perfect Forward Security and Post-Compromise Security
uses X3DH (extended diffie hellman)
to initialize Double Ratchets (asymmetric / symmetric ratcheting)

11.1 server

handles asynchronous message delivery
delivers when parties come online

11.2 threat model

completely controlled network (Dolev-Yao)
reveal derived message keys
corrupt long-term secrets
compromise ephemeral (one-time use) secrets

11.3 properties

signal archives both notions simultaniously

perfect forward security

attacker corrups long-term key of A
can now impersonate A, but not learn old messages
key indistinguishability of previous keys holds

post-compromise security
attacker corrupts and compromises full state of A,B
then attacker becomes passive
security is recovered after A,B executed without modification

11.4 protocol stages

might overlap

keys
long-term Identitiy key (id)
med-term Signed Prekey (sp)
ephemeral One-time key (ot)
each diffie hellman public key pair

registration phase
PreKey bundle = (id, keys, signature)
id long-term identifier like phone number
keys as described above
signature uses id (long-term) to sign sp (med-term)
PreKey bundle uploaded to public server
but id not bound to signature / keys (needs offline validation)

initialization phase (X3DH)
party retrieves PreKey bundle of other party
combines keys using X3DH to get pre-master secret
creates first (asymmetric) ratched key
first root key through KDF of premaster & ratched key

X3DH combinations
given public sp_A, id_A, ot_A (optional) of other party
given secret sp_B, ot_B of self
ot_A with ot_B for forward secrecy
sp_A with ot_B prevents KCI against A
id_A with ot_B prevents KCI against B
sp_A with id_B to auth B, id_A with ot_B to auth A
no id_A with id_B for deniability

asymmetric ratchet phase
assume that A&B have shared root key rk_i
A&B additionally agree on s = g^{xy} (evolving DH secret key)
A → B g^{a_i}, B → A g^{b_i}, A → B $g^{a_{i+1}}$, ...
then get key sk$_{i+1}$ = KDF(rk_i, s)

symmetric ratchet phase
next messages within flow of messages
assumed that A&B have shared symmetric key ck
use "ratched" for each new message (ck$_{i+1}$ = KDF(ck_i))
offers perfect forward secrecy
as whenever ck_i compromised, previous ck_j not exposed

double ratched
combination of asymmetric & symmetric ratched
asymmetric ratched with each message exchanged (ping-pong)
symmetric ratched for successive messages (without exchange)

message encryption
AEAD encryption with authenticated data AD
rc_A for most recent public ratched key
id_A, id_B for public key identifier of A, B
PN for #messages in last chain (to know when to remove old keys)
ctr for message index in current chain (for out-of-order decryption)
AD = rc_A || id_A || id_B || PN || ctr

12 message layer security (MLS)

create group message service out of two-party messaging protocol

12.1 binary tree

each node two children
root note is top-most node
leaf nodes are nodes without children
decendants are children (recursively)

paths
directed from C (path C until node)
co-path from C (other children on path until node)
like hashes required by mercle-hash tree

tree structures
full (if 2^n nodes distributed)
balanced (either root balanced, or left child largest full subtree)

12.2 naive approaches

pairwise channel (client-side fan-out)
naive approach creating O(n^2) pairwise channels
linear cost in size of group (ratcheting operations)

reducing overhead (server-side fan-out)
new members generate new ck_0, keypairs & distribute to group
new message derives new ck_i (all members derive same)
constant cost, size of group irrelevant
no post-compromise security, no deniability

12.3 MLS protocol RFC

IETF workgroup, acamedia & industry

setting
federated E2E secure group messageing
support large groups (50K) with low bandwidth/complexity
asynchronous, long-lasting sessions
dynamic group membership

security
E2E authenticity and privacy
network adversary (Dolev-Yao), active, corrupting state
Perfect Forward Security and Post-Compromise Security

12.4 general setting

authentication service (AS) for trusted identity-key mapping
delivery service (DS) to route key material & messages in order

setup
members create accounts & get credentials from AS
members authenticate to DS & store Key Package
when party sends message, retrieves Key Packages from DS
then uses key material to create c which is distributed by DS

message delivery
reliable (all messages are delivered)
in-order (as recieved by DS, sent to clients)
consistent (all clients have same view of messages & ordering)

group policies
any member can add or remove others
restrictions responsiblity of application layer

12.5 ratched tree

group members arranged at leaves
each node has public keypair associated
each member knows secret keys on direct paths
each member knows at least public keys on co-path
root node has commit secret associated

key derivation
key valid within epoch i
any member can advance epoch by updating commit key k
st_i = KDF(sk$_{i-1}$, k_i) for sk ratched key
besides sk_i, multiple keys for different purposes retrieved
combination with sk$_{i-1}$ gives post-compromise security

update commit key sk
member creates new secret key
uses KDF to generate new keypairs on path up until root
on co-path, encrypt under their public key these new secrets
DS distributes updates to all users
$log_{2(N)}$ scaling (bandwith = len(ctxt), computational = KDF + PK)

blank nodes
when adding/removing members, some nodes get blanked
meaning no secret/public key is assigned
when on direct update path, all blank nodes get new keypair
when on co-path, instead encryption under children's public key

add new member
receives joiner secret encrypted under Key Package public key
using joiner secret, can perform an Update
when full tree, new root (left = previous root, right = new member)
else, replace existing member with three-node subtree

remove member
blank all nodes on direct path
no longer encrypt new secrets on path to removed member

12.6 secret tree

identical structure to ratched tree
to turn derived encryption secret k_{es}
into secret for node for message encryption ts_i

master secret derivation
depth-first numbering of nodes
start at root, recursively KDF with node index to get to leaves
ts_2 = KDF(k_{es} || 2), ts_3 = KDF(ts_2 || 3) (for node 2 parent of node 3)

deriving encryption key
using secret of specific node n initiate symmetric hash rachet
derive sequence of single-use key & nonces
ratched forward for each message

MLS ciphertext
groupID, epoch, contenttype
authenticated data & encrypted sender data
AEAD using derived nonce, single-use key
handshakes additionally MACed

12.7 security

correctness (same commit secret k in each epoch)
privacy (k looks random given transcript of messages)
forward secrecy (state leak does not reveal previous k)
post-compromise secrecy (after update, k becomes secret again)

discussion
ratched tree well designed (and only log n runtime)
but secret tree complex (deriving keys is easier solvable)

13 fully homomophic encryption (FHE)

13.1 history

since 1978 partially homomophic schemes
since 2000 more practical, but still partial
since 2010 fully homomophic scheme proposed
1st generation systems required 30min for single computation
2nd generation with less expressiveness, seconds for computations
3rd generation with miliseconds for computation

13.2 challenges

crypto (underlying math, parameter selection, security)
computation model (no if/else, no loops/jumps)

13.3 construction

conceptually simple, but based on difficult crypto
based on relatively new hardness assumptions

(ring-)learning with errors assumption (RLWE)
hard to find s, given c and a
for c = a∗s + e
like R = Z[x]/$(X^n + 1)$

enc/dec
for μ = m ∗ q/t, random e, a, s
enc(m) = (c_0 = a ∗ s + μ + e, c_1 = a)
dec((c_0, c_1)) = t/q (c_0 - c_1 ∗ s)
results in m as long as e is small

operations
multiplication has 1/q due to scaling factor
add(c, c') = (c_0 + $c_{0'}$, c_1 + $c_{1'}$)
mult(c, c') = 1/q ∗ ($c_0 * c_{0'}$, $c_0 * c_{1'} + c_1 * c_{0'}$, $c_1 * c_{1'}$)
relin(c_0, c_1, c_2) = (c_0, c_1) (using bootstrapping)

bootstrapping
to reduces error level ("relinearize")
encrypts under new K, reduces error level, decrypts again
most expensive computation (order of seconds, minutes)
the major breakthrough in 2009

speed
in general, seconds on laptop → minutes of server
if stupidly parallizable, then 2-3x slower
if complex computation, then arbitrarily slow

13.4 building system

getting it secure & fast difficult

parameter selection
small as possible for efficiency
but large enough for security & correctness
for n polynomial size, q coefficient bitwidth (noise overflow)
security improves with larger n and smaller q

13.5 programming paradigm

algorithm & input bounds are known, but data is secret
no branching depending on secrets allowed (would leak bit)
easy SIMD target so performance does not suffer that much

no if/else
compute both branches
then multiply such that only result remains
like c ∗ b_0 + $(c - 1) * b_1$ for c comparison, b_i branches

no loops
unroll completely according to worst case assumption
then use comparison trick so only useful b_i remains

13.6 compilers

map high-level programs to arithmetic circuits supported by FHE
replace computations, but also order for performance
exist at different abstraction levels
EVA fast for general purpose algorithms
nGraph fast for machine learning

14 post-quantum cryptography (PQC)

conventional public-key cryptosystems resisting quantum algos
hence use classical computers, but quantum-safe assumptions

14.1 quantum-safe assumptions (for classical computers)

lattice based (learning with errors, ...)
code-based (McElice encryption, ...)
non-linear system of equations
elliptic courve isogenies

14.2 quantum concepts

schroedingers cat
strict interpretation of quantum physics
radioactive source releasing poison in box with cat
until box opened, cat both dead and alive

qubit
basic unit of quantum computation
superposition of "1" and "0" bits

quantum computing
executing sequence of quantum gates (like AND, OR classical)
all possible states input → all possible states output
hence can compute all classical states in parallel

14.3 progress

few progress 2003 - 2015 (few qubits)
2019 google sycamore chip with 53 qubits, 0.1% errors

error correction
below some error / above some number of qubits
then can perform (perfect) quantum error correction
for 0.1%, around 1000 qubits for 1 correct qubit required
hence >100 mio qubits required for shor's algorithm

sycamore chip (2019)
53 qubits, operating at 20mK
organized in grid, each qubit connected to 4 neighbours
evaluated with random circuits (not useful, but hard to simualte)
quantum supremacy claimed (quantum computer faster than classical)
but IBM contradicted claims, fast simulation was shown

timeline large-scale quantum computing
hyped field, lots of research investment & smart people involved
hard to tell if breakthrough imminent, moores law, ...
IBM claims 1000 qubits by 2022, moores law applies
but others claim it a ponzi scheme to get funding

14.4 core algorithms

shor's algorithm (1994)

finds period of a long sequence
number of qubits / circuit depth polynomial in input size
reasonable constant / value of polynomial
efficiently solves integer factorization problem (IFP)
efficiently solves discrete logarithm problem (DLP)

grover's algorithm (1996)
can perform fast unstructured search problems
but requires very deep quantum circuit
quadratic speed up (optimal in general case)
solves O(2^{128}) keys with O(2^{64}) sequential AES
symmetric algorithms can simply double key size

14.5 time is of essence

switch has to happen before large-scale quantum computer available
but current data has sensitivity lifetime ("store now, decrypt later")
cover time of crypto methods has to exeed sensitivity lifetime
development progress observable as done by big public firms

transitional risks (theoretical)
young field (sudden drastic progress possible)
like with runtime of alogs, hardness assumptions
provable security only with large factors
but code-based assumption seems stable

transitional risks (practical)
implementation vulnerabilities (newly written, incomplete code)
early lock-in of bad choices through premature deployment

14.6 NIST POC standardization algorithms

to publish cryptographic standard for quantum algorithm
portfolio of choices depending on application will be selected

history
2012 formal start
2016 formal call for submission 2017
2017 69 "complete and proper" submissions
2019 26 candidates
2020 7 finalist + 8 alternatives
2022 planned standard publication

types of algorithms
signatures
public key encryption for symmetric key transport
key-establishment KEM (confusingly worded as diffie hellman)

submission requirements
included guidance on security proofs & resource measurement
number of classical elementary operations & circuit size
should consider realistic circuit depth limitation (2^{40} gates)
parameters for at least some of the 5 different security levels

progress
first round w/ 69 candidates
second round w/ 17 public key, 9 signature schemes remaining
final round w/ 4 public key, 3 signatures remaining
with 8 "alternate" candidates remaining, to be standardized later

comparison
smaller ciphertext / public key sizes depending on category
structured lattice < quasi-cyclic code < unstructured lattice
isogeny elliptic course with smalles cipher/public key, but slow
Classic McElice small ciphers, but huge public keys (300k bytes)
ECIES beats all schemes → capability of quantum computation?

14.7 Classic McElice

fast algorithms (KGen, Dec, Enc), compact ciphertext
coding-based KEM with tight security proof
but large public key size (300kb at 128bit security level)

history
1978 invented (close to RSA invention)
1986 crucial simplifications by niederreiter

security assumption
public key looks like random error correcting code
stable assumption (fastest set decoding algorithm of 1962)

[n, k, d] linear binary code
for G be k × n binary matrix (= in F_2) of rank k
code C is linear combination of rows of G
C has minimum distance d iff min {$d_{hamming}$(u,v) for all rows u,v}
for $d_{hamming}$(a,b) counting in which bits combination a differs of b
can correct up to t = (d-1)/2 bit-flips in C

example reed-mueller code
[2^m, m+1, 2^{m+1}] for integer m
for m = 3, get 4 × 8 matrix G
encode x (for |x| = m+1) with xG
decode xG + e using Fast Hadamard Transform
given e has hamming weight at most 1 = (4-3)/2 = (d-1)/2

core idea McElice
let G be generator matrix of [n,k,d] linear binary code
let S be random invertible k × k, let P permutation n × n
let G' = SGP with G' looking like random k × n matrix of rank k
(requires appropriate choices of G to indeed look random)
given c = xG' + e (with hamming weight e ≤ t), finding x should be hard
knowing S & P, decoding easy of c' = c ∗ P^{-1} = (xS)G + eP^{-1}

McElice PKE scheme
KGen chooses S, P and uses static G
returns sk = (G, S, P), pk = G' = SGP
Enc(pk, m) chooses e (unif. random w/ hamming weight ≤ t)
returns c = mG' + e
Dec(sk, c) computes c' = cP^{-1}
runs decoder on c' to get m', e'
returns m = $m'S^{-1}$

Classic McElice
uses Niederreiter adaption for enc/dec
replace G' with H such that $G'H^T = 0$
Enc(pk, m) returns c = mG' + e (for G' now H)
Dec(sk, c) can extract e with c ∗ H^T = e ∗ H^T

further design choices
chooses Goppa codes as linear binary codes
decoding possible using berklekamp-massey algo
IND-CCA by adding hash to ciphertexts