Information Security

27110 characters in 4899 words on 735 lines

Florian Moser

August 8, 2018

1 introduction

cryptography denotes the construction of secure systems
cryptoanalysis is used to break said systems
cryptology is cryptography + cryptoanalysis
used to be art with ad-hock design, usually insecure, arms race
now science with formal definitions, systematic designs & constructions

1.1 provable security

motivation
can't just simulate with typical input
because attacker won't respect "typical input"

kerkhoff's principle
enemy knows the system except a short key k (chosen at random)
because unrealistic to assume secrets (reverse engineering)
because short keys easy to generate, protect, replace
because design details can be analyzed/discussed publicly

physically unclonable functions (PUF)
attacker cannot replicate design

mathematical view
key space K, plaintext space M, ciphertext space C
encryption scheme is pair (Enc, Dec, (Gen?))
KxM → C is encryption
KxC → M is decryption
correct if for all k Dec(Enc(m)) = m

1.2 historical ciphers

shift cipher (ceasar)
letters in secret are shifted by k
break with brute-force, statistics

substitution cipher
letters in secret are shifted by K[letter]
generalization of ceasars
break with statistics (d-time frequency attack)

vigenere cipher
letters in secret are shifted by K[pos % len(K)]
if keysize d known, break with d-time frequency attack
find identical l-grams, take gcd of their distances d_i
do d-time frequency attack on chars at distance d_i

1.3 information-theoretic security

define meaning of security
construct schemes that are provable secure

bad definitions
"k uncomputable" but then simply don't encrypt
"m uncomputable" but maybe parts of m computable
"learn nothing of m" but maybe already has some knowledge m

definition
adversary can not learn any additional information about m

perfect security
P(M=m) = P(M=m|C=c)
requires the distribution of M,C to be independent
Enc(k,m_0) same distribution as Enc(k, m_1)

optimality (Shannons theorem)
k ≥ c because else one k could map to same ciphertext
c ≥ m because m→c is an injection
therefore k ≥ m
necessary but not sufficient for perfect security

1.4 one-time pad

perfectly secure
k XOR m = c

proof
P(C = c | M = m) = P(M XOR K = c | M = m) =
P(m XOR K = c) = P(K = c XOR m) = 2^-t

generalization
M, K, C all equal to group G, k with inverse
Enc(k, m) = m ~ k, Dec(k, c) = c ~ k^-1

not practicable
key has to be as long as message
key cannot be reused
key requires a lot of randomness because long

1.5 unconditional security

one-time pad
quantum cryptography

1.6 computational security

assume some problems are computationally difficult
assume understanding of computationally difficult is correct

probabilistic TM (PTM)
like TM, but additional tape consisting of random bits
output random variable M(X) for input X

negligible function (nf)
if for all c ∈ N, n_0 for chosen x > than some n_0
nf(x) < 1/x^c for all natural numbers c
negligible if like 2^-n, n^-n
not negligible if like n^-2, n^-1000

security parameter
1^n, denoting length of secret key
guessing key is negligible because probability of 2^-n
but enumerating keys possible because in time 2^n

(t,e) secure system X
every TM operating in time t (=efficient computation)
can break X with probability at most e (=very small)

asymptotic approach for (t,e)
describe (t,e) using asymptotic notation depending on parameter n
polynomial-time computable (O(n^c)) on a probabilistic TM
good because no need for more details (church-turing assumption)
bad because need to reason informally about concrete systems

in practice
formally prove asymptotic result
argue informally that constants are reasonable

1.7 security definitions

to prove security definitions construct games
fulfilled if P(correct) = 0.5 + e(n) for negligible e(n)

perfect secrect encryption
no poly-time adversary can distingush Enc(m1) and Enc(m2)
with non-negligible probability
it holds that |K| ≥ |M| (by shannons optimality)
it holds that P[M=a|C=c] = P[M=a] (by perfect secrecy definition)

indistinguishable encryption (IND)
adversary chooses m_0, m_1 of same length, send to oracle
oracle selects one & encrypts
adversary needs to guess which message was encrypted
ensures that distributions Enc(k, m_0) = Enc(k, m_1)
implies that encryption must be perfectly secret
called IND-security, semantic security

known plaintext attack
know of some plaintexts, but the adversary does not control which

then IND game

batch chosen plaintext attack
can choose some plains to be encrypted by oracle
has to be chosen for a single time, at once
then IND game

adaptive chosen plaintext attack (CPA)
adversary repeatedly chooses plain & oracle encrypts (learning phase)
then IND game, may uses same plains as in learning (challenge phase)
ensures that semantically secure even if with chosen plaintexts
implies that |message| must be bounded (else break with |m0|=1, |m1|=p(n)+1)
implies that encryptions have to be randomized/stateful (else learn m1)

chosen ciphertext attack (CCA)
either before or after the challenge is received
adversary can ask ciphertext != challenge to be decrypted by oracle

2 symmetric cryptography

2.1 proofs

proofing properties
if <computational assumption> then ... (like DDH, RSA)
if <some schema A secure> then ... (like SSH)

IND-CPA (for k < m) implies P!=NP (reverse unknown)
for all (c,m) there exists a key k such that Enc(k, m) = c
L is in NP because k is NP-witness, therefore assuming P=NP
create TM which decides for (c,m) if in L (so if k exists)
ask TM with (m_0, c) in IND game
p non-negligible that answer correct ($\geq 3/4$)
because if oracle encrypted m_0 then TM answers always correctly
TM may fails if m_1 can be encrypted to same c (with different k)
which happens with $p \leq 1/2$ for keysspace=4 / messagespace=8

PRG implies secure encryption
using a reduction of the IND security game g
adversary replaces key of game with random or PRG-output
if g succeeds, output "pseudorandom", else output "random"
assuming RPG not secure, then "pseudorandom" p=0.5+e(n)
therefore |0.5 - (0.5 + e(n))| = e(n) which is not negligible

secure encryption implies one-way function
construct one-way function using encryption with m = 0
because key (= input one-way function) needs to be well distributed

one-way function implies PRG
proof ommitted

one way function implies NP!=P
because poly-time computable (NP witness)

2.2 pseudorandom generators (PRG)

2.2.1 requirements

looks random
for TM D, random R, short random S, PRG G
if |(P(D(R))=1) - P(D(G(S)))=1)| negligible in n

expands input by some factor l
seed s as input, G(s) as output
length G(s) = l(n) for l expansion factor

2.2.2 golomb's postulates

G1 (balance property)
occurrences of 1 and 0 should be about the same
difference should be smaller, equal 1

G2 (run property)
runs (consecutive 0s or 1s) of length i occur with p=2^-i
the same number of 1 and 0 runs exist for all lengths i

G3 (correlation property)
occurrence of 1 following 0 (& vice versa) equals some c
c must be equal for all sequences s = G(s) >> i for all i

2.2.3 constructions

approaches
using one-way functions (theoretical result, inefficient)
using one-way permutation with hardcode bit
based on hardness of problem (impractical)
based on stream ciphers (bit juggling, practical but informal)

linear feedback shift register (LFSR)
passes golombs postulates, but output predictable after 2n
register with n bits, outputs position n, insert new at position 0
inserted bit computed over all bits in register (XOR)

blum blum shub PRG
assumes factoring large n difficult
select primes (p,q) such that p,q mod 4 = 3, let n = p*q
select seed s < n that gcd(s, n) = 1 (coprime)
output LSB for each x_i = x_(i-1)^2 mod n for x_0 = s
but will repeat after some time

PRG derivations of existing PRGs
prove that requirements are still met (expansion, randomness)
if secret expanded then construct PRG only using expansion
if secret shortened then check if still expansion happening
if same secret reused, will result in same output!

using one-way function
theoretical result, proof ommitted

using permutation
need a hardcore bit (bit which breaks some hardness assumption)
for example XOR over all values for one; gives l=|x|+1

2.3 one-way functions

2.3.1 requirements

poly-time computable
therefore only exists if P!=NP (because of NP witness)

hard to invert
for random x, y = f(x); if (A(y) = x' $\Rightarrow$ f(x') = y) = e(n) negligible
not required to hide all input (therefore unfit for secrecy)

2.4 pseudorandom permutation (PRP)

true permutation can be implemented with shared lookup table

2.4.1 requirements

permutation
domain/range is of the same size
for all x F(x) is a bijection

poly-time computable
for all x F(x) and F^-1(x)
distinguisher can't distinguish true random or PRP

indistinguishable
distinguisher can't distinguish from random permutation
strong PRP if distinguisher can also ask for F^-1

2.4.2 keyed PRP

use a key

permutation
for all k F(k, x) is a bijection

poly-time computable
for all k F(k, x) and F^-1(k, x)

2.4.3 pseudorandom functions (PRF)

like PRP, but without permutation requirement
hence output may be of different size than input

2.5 stream ciphers

PRG in practice
output is infinite stream of bits

requirement
after some start_position has to look random for random seed s

synchronized
produce bit steam with G(s), need to sync state with decryptor

unsynchronized
bit stream with G(IV, s) for plain transmitted or known IV
need better G to be secure even if adversary knows IV
construct from G(s) with G(hash(s || IV)) or design from scratch

RC4
very efficient & simple, but some flaws
no IV, some biased output, some leakage in first bites
TLS problems include k only 40bits, IV reset/changed often
WEP problems include insecure, blacklisted IV's
WPA uses longer IV & key

2.6 block ciphers

PRP in practice
often better choice than stream ciphers
output is block-wise stream of bits
can emulate stream cipher with counter mode (if block size big)

2.6.1 requirement

x = G(k, block_content) has to look random if K is random & secret
x can be inverted knowing key without any other information
(stream cipher additionally needs to know position)

2.6.2 modes of operations

PRF with key k, F, message
in blocks m_i, random in blocks r_i
prove with F_k replaced by truly random for distinguisher

why
encrypt long messages (longer than block size)
introduce CPA security (which needs state)

possible properties
provable secure (0)
no error propagation (1)
parallel encryption (2)
parallel decryption (3)
recompute only single block if bit in plaintext changed (4)

naive
c = r1, F(r_1) XOR m_1, r2, F(r_2) XOR m_2
but needs a lot of randomness, 2n expansion

electronic code block (ECB)
c_i = F(m_i)
no (0) because of CPA, yes (1) (2) (3) (4)

cipher block chaining (CBC)
c_i = F(m_i XOR c_i-1), c_0 is IV
yes (0) (1) (3), no (2) (4)
not CCA (send c,IV=0 to oracle, then XOR with IV to get m0/m1)

output feedback mode (OFM)
c_i = F(c_i-1) as PRG, c_0 is IV; c XOR m
yes (0) (1); (2) (3) if stream precomputed, (4) leaks info

counter mode (CTR)
c_i = F(IV + i) as PRG; c XOR m (need high block size)
yes (0) (1) (2) (3), (4) leaks info
not CCA if IV !rand && incr (m0=01; then encrypt m1=00)

2.6.3 padding

if m % b != 0 need to pad to reach block size

PKCS#5/7 padding
fill the x padded bytes with the value x
to remove padding; read value in last byte & remove
but if m%b=0 need full additional block

CBC cipher text stealing
pad with q zeros to match block size
encrypt m_n-1, split at q, first part=c_n
m_n XOR c_i-1, q last bits replaced by second part, =c_n-1
start to decrypt c_n, take last q bits and add to c_n-1
decrypt c_n-1, finish with XOR of c_n, remove 0 padding

2.6.4 shannon principles

confusion (key bit change influences whole ciphertext)
diffusion (pbit i changes cbit j with p=0.5)

implement by iteration
confusion using large PRP of block size b
diffusion by permuting (mixing) output (overcome b limitation)
repeat so changes in input can propagate
formalized by feistel networks

implement with product ciphers
increase security of PRF with F(F(m)) for different keys k
only hardens if F not idempotent (so no F(m) = F(F(m)))

2.6.5 building blocks

substitution-permutation network
may XOR input first with key/subkey
confusion with s-boxes (change $\geq$2 bits for 1 bit input change)
diffusion with static permutation (spread s-box output)
specification public, so can invert single rounds easily

feistel network
needs key schedule, permutation f (inversion not needed)
split input into left L, right R
(1) left = left XOR f(key, right)
(2) then switch left/right & repeat at (1)
for easy inversion omit (2) in last round
then just reverse key schedule to invert
3-round for PRP, 4-round for strong PRP
changes cbit with p=0.5 after 8 rounds for pbit changed

2.6.6 DES

2^56 key space, 2^64 block size
input through initial permutation
then 16-rounds feistel network with 48bit subkeys
output put through final permutation

function f
expands input from 32bit to 48bit (duplicates some bits)
XOR with subkey
confusion with 8 s-boxes mapping 48bit to 32bits
diffusion with static permutation

security
developed with NSA, but no backdoors found (yet)
design decisions taken to harden differential cryptoanalysis
but short key & block size

increase key size
cascade (product) the ciphers

triple encryption
execute decryption with second key
called 3DES, but rather slow & still small block size

break DES rounds
need input/output pairs
can calculate feistel network variables L_0, R_0, etc
then inverse f (inverse permutation, s-boxes)
calculate s-box possible values (4^8)

bruteforce double-DES
need input/output pairs
for all k1 encrypt input, for all k2 decrypt output
get 1/2^64 match probability; for 2^56^2 keys
therefore get 2^48 matches (k1, k2)
use keys for next input/output pair to reduce p by 2^64

break enhanced DES schemes
need input/output pairs
reduce to-be-found keys by showing some to be deducable
find candidate keys by decr/enc and comparing results

2.6.7 AES

chosen because won public competition
recent hardware has dedicated assembly instructions

substitution-permutation network details
state = input XOR key, get matrix in 4x4 form
SubBytes replaces byte with lookup table entry (confusion)
ShiftRow shifts by 0 in 0 row, 1 in 1st row, ... (diffusion)
MixColumns multiplies with matrix in GF(2^8) (skipped in last round)
state XOR key, repeat at SubBytes for 9 rounds more

3 hash function & macs

3.1 MAC

for message authentication
attacker not be able to compute tag for new messages

mathematical
Vrfy(k, m, Tag(k, m)) = yes for key k

security
adversary can choose m_i, and asks oracle to Tag it
but can't produce m', t such that Vrfy(k, m', t) = yes

properties
existential forgery if attacker can generate one such pair
universal forgery if attacker can generate t for all m

3.2 block cipher MAC

define Tag(k,m) = F(k,m)

bad ideas with blocks
authenticate separately (could reorder)
add counter (could cut off)
add length/counter (could cherrypick parts of messages)

naive implementation
add message-random & length & counter to each block (4 times size!)
tag = message random || MAC(message || 0 padding)
secure (assuming not, can show that F different from random)

CBC-MAC
produce tag with CBC-mode & IV = message length
need IV, else m'=m1||(m2 XOR t1) with valid t'=t2 for given m1,t1,m2,t2
do not publish intermediates

CBC-MAC with 2 keys
F(k1, CBC-ENC(k2, m))

3.3 hash functions

3.3.1 requirements

poly-time algorithm
hence needs P!=NP
H(s={0,1}^n, x={0,1}^*) = {0,1}^l(n) for l(n) fixed function

collision resistance
oracle selects random s to determine H (else could precompute collisions)
adversary tries to find m,m' such that H^s(m) = H^s(m')

forms of collision resistance (weak to strong)
preimage (given v, find x such that h(x) = v)
weak/2nd preimage (given x, find x' != x such that h(x) = h(x'))
strong (can't find m, m' such that H(m) = H(m'))

3.3.2 modular construction

compression function
fixed-length, collision-resistant function h : 2L → L
naive (XOR (IV + message + 0 padding)) but can produce m' = m || 0
merkle damgard transform (XOR (IV + message + 0padding + length))
use a (strong) pseudorandom permutation such as CPA-secure encryption

H and h collisions
collision in H implies collisions in h
no collisions in h implies none in H
for |m|=|m'| there must be different blocks hashing to same result
for |m| != |m'| it must be at least the last one (bc different lengths)

attack for H(k||m) for merkle damgard
as the last block of size b is length of message l
can append another last block with value l+b

NMAC
H(k2, H(k1, m)), secure for h collision resistant & H(k2, m) secure MAC
but key too long, hash functions can't change IV

HMAC
H((k XOR opad) || H((k XOR ipad) || m)) for ipad=0x36, opad=0x5C
pads maximise hamming distance, double hashing prevents extension

3.3.3 applications

authenticate long messages
F(k, H^s(m)); need only single encryption step
used in pk cryptography as "hash-and-sign"
prove by assuming adversary can break a tag
then show that it can distinguish F or break H

uniform randomness generation
using nonuniform source (key strokes, passwords)

4 discrete math foundations

4.1 group definitions

group
has identity element (1)
closed under operation (g^a ∗ g^b = g^a+b mod p)
every element has an inverse (1 = g^a ∗ g^a-p)

abelian
associativity, commutativity (order of operations does not matter)

ciclic
if element exists which generates whole group (called generator g)
if p prime, then Z_p always cyclic
if |G| prime, then all numbers except 1 generators

order
size of group
subgroup size generated by element (divides group order)

4.2 group proofs

g^m = 1
write g1∗g2 = (gg1)∗(gg2) = g^m(g1∗g2)
because m different (gg1) therefore all unique

4.3 discrete log

generator produces permutation of elements in G
finding out x = log_g(y = g^x) in group Z_p∗ is believed to be hard

efficient exponentiation
create g^1, g^2, g^4, ..., g^i < g^x
multiply all g^i where log2(x) position not 0

discrete log assumption
H(1^n) = (G, g) for G group of order n & a generator g
oracle gets (G, g)=H(1^n) for 1^n security parameter
adversary gets (G, g, y) and needs to compute x = log(y)
problem hard if P(A can compute)=e(n) is negligible in n
therefore f(x)=g^x behaves as one-way function

parity problem (even/odd)
QR = all a where a = b^2 mod p for some b (quadratic residue modulo)
QR subgroup of Z_p∗ because (1, closed under multiplication, has inverse)
QR = {g^2i} for all i (even numbers); |QR| = |Z| / 2 = (p-1)/2;
test a to be in QR with a^(p-1)/2 mod p = 1 (because a = g^2i)

mitigate parity problem
use QR_p instead of Z_p as group, for safe prime p = 2q + 1
hence QR has prime order q, therefore all elements generators

square root in QR
sqrt(x) = x^m+1 for 2m + 1 = order QR_p
there are proofs for p = 3 mod 4 and p = 1 mod 4 (therefore for all)
for p = 3 mod 4; can write p = 4m + 3; observe that |QR| = 2m + 1
claim sqrt(x) = x^(m+1) and show with sqrt(x)^2

4.4 Zn∗

Zn∗ = {a ∈ Zn : gcd(a,N) = 1} (is an abelian group)

finding inverse
use extended euclidean algorithm (EEA)
for input a,b gives x,y,z such that z=a∗x + b∗y
if gcd(a, o)=1=z then x is inverse of a in mod o

finding e-th root
equals finding inverse
because for f(x) = x^e mod N
f^-1(y) = y^d mod N for d inverse of e

4.5 chinese remainder theorem

for n_i pairwise coprime (gcd(n_i, n_j) = 1), for a_i integers
given that x = a_i mod n_i, N = sumof n_i
then there exists unique x 0 < x < N

5 public key cryptography

does not need authentic/confidential channel for keys
supports non-repudation
but slower to compute

5.1 key distribution

predeploy pairwise keys
but very naive, quadratic number of keys required

key distribution center
server S gives keys, need shared key with S
but S single point of trust/failure

pk register
public key pk kept in public register, anyone may encrypt/verify
secret key sk stays locally, owner may decrypt/sign

identity based pk
encrypt directly with ID as public key
server S can generate secret key for ID
no need for PKI, but S single point of trust/failure

5.2 construct sk/pk schemes

can be based on famous mathematical conjectures
constructions have mathematical structure (lego)
constructions have natural security parameter (the key)
sk serves as trap door (easy computation of hard problem)
but not efficient, structure may helps with breaking

formal
(Gen, Enc, Dec) triplet
Gen generates (sk,pk) keypair
Enc takes pk, m and outputs c
Dec takes sk, c and outputs m

5.3 applications

signatures
publicly verifiable (can prove to third party)
transferable (can prove to n parties)
provide non-repudiation (signature binding)
but who maintains register, how to CRUD securely

hybrid encryption
encrypt symmetric key with asymmetric crypto for long messages
more performant but resulting in same guarantees

5.4 diffie hellman key exchange

with public hard discrete log group G, generator g
$A \to B$: g^x; $B \to A$: g^y; A&B can calculate k=g^xy

secure key exchange
if |(P(A(1^n, k))=1) - P(A(1^n, r))=1)| negligible in n
so A can't differentiate key from random of same length

decisional diffie hellmann (DDH)
static input si = {G, g, p, g^x, g^y}
if |(P(A(si, g^z))=1) - P(A(si, g^xy))=1)| for random z
but g^xy in QR with p=3/4 (odd∗even), g^z with p=0.5
DDH does not hold in Z_p∗, but in QR_p

generate groups for DDH
choose prime q, p = q∗2 + 1 such that p > n+1 for 1^n
choose x ∈ Z_p for such that x != 1, x != -1
set g = x^2 mod p (to get generator from QR_p)
output (x, g) for (generator, group)

5.5 elgamal encryption

$A \to B$: public key (G, g, p, h=g^x)
$B \to A$: h2 = g^y, c = m XOR h^y
A can decrypt with c XOR h2^x
needs fresh secret for each usage
can use h2 directly for hybrid encryption
can be generalized to other groups (like elliptic curves)

5.6 RSA

choose primes p,q, let n = p∗q
let o = (p-1)(q-1) = phi(N)
choose r relatively prime to o (gcd(o, r) = 1)
compute d = e^-1 mod o (d∗e mod o = 1) with smaller(EEA(r, o))
pk (e, n), sk (d, n)
c = m^e mod n for encryption, m = c^d mod n for decryption

correctness
m = m^ed = m^(1+k∗o) for some k (because d∗e mod o = 1)
= m(1)^k (because m^o mod n = 1 because of eulers theorem)

finding o equally hard to factoring N
if q,p known then o easy to find
if N,o known, then p,q easy to find with system of equations

finding d from (e,N) equally hard than factoring N
proof ommitted

e-th root
same as computing d, because d∗e mod o = 1
efficiently with EEA(e, o) = de + ko = 1

RSA assumption
computing e-th root is hard without o
c = m^e mod N and c ∈ Zn∗, e ∈ Zo∗ with gcd(e, o) = 1
P(A(c, N, e) = m) = e(k) negligible in k for

RSA over Zn
some elements not in Zn∗, but hard to find
assume x mod q = 0, then gcd(x, N) = q (so factoring N easy)
x still works with RSA, because of chinese remainder theorem
x^ed = x (mod q) (holds because x mod q = 0)
x^ed = x (mod p) (holds because of d construction)

textbook weaknesses
homomorphic; (RSA(m0∗m1) = (m0∗m1)^e = RSA(m0)∗RSA(m1)
deterministic; can compute c0=RSA(m0), compare with c from oracle

CCA attack
intercept c, encrypt c' = r^e ∗ c for arbitrary r
send to decryptor to get m', get m = m ∗ r^-1

same N attack
if e,d,N known can deduce o with ed = x∗o + 1
therefore crack d of others using same N,o

PKCS#1 padding
create random of size s = k (length N) - D (size plain) - 3
padding like like 0-byte | 2-byte | random | 0-byte | plain

chosen ciphertext attack padding
maybe possible if padding position changes with |message|
assume padding = 0^k | random | 0000 for k variable
query oracle with m1=0000/m2=01111, get c
multiply c by 2^e, resulting in left shift
ask decryptor; if no error can determine m1 or m2 by MSB

6 zero knowledge proofs of knowledge

6.1 properties

for prover P, secret s, verifier V
every NP problem can be reduced to circuit SAT (assignment proof)

6.1.1 completeness

protocol succeeds with high probability for honest P,V

6.1.2 soundness

no wrong P can convince honest V with non-negligible p
to prove, show that knowledge extractor exists

knowledge extractor
given P, can compute the secret
has power to rewind P to previous state
need to choose all values random
else honest V could include special cases for specific c

6.1.3 zero-knowledge

the prover leaks no information about s
to prove, show that simulator exists

simulator
forges a transcript (is therefore able to choose challenges)
honest V can't distinguish forged from real transcript
need to argue that all values are random or derived from random
therefore distribution of transcript like real distribution

indistinguishable
perfect (for unbounded impossible, can generate exact replica)
statistical (unbounded attacker has negligible)
computational (bounded (polynomial) with negligible)

6.2 ZK cave example

cave with door at the bottom, left & right shaft
verifier checks if door is locked
prover goes into cave without verifier looking
verifier tells prover to use left, right shaft
verifier can cheat with 50%

6.3 Fiat-Shamir

prove knowledge of square roots in RSA group

public parameter
n = pq for large primes

knowledge of P
t = s^2 mod n
t is also know to V, but without s

protocol
P chooses random r in Z_n∗
$P \to V$: x = r^2 mod n

V chooses c = {0,1}
V → P: c
P → V: y = r*s^c mod n
V verifies that y^2 = x * t^c mod n

soundness
send c1 = 1, get y1 = r*s,
rewind and send c2 = 0, get y2 = r
s = y1 * y2^-1

zero knowledge
for c=0 pick r, print x = r^2, y = r, c
for c=1 pick y, print x = y^2 * t^-1, y, c

6.4 Schnorr

prove knowledge of discrete log

public parameters
G with hard discrete log & generator g

knowledge of P
t = g^s
t is known to V too (but without s)

protocol
P chooses random r
P → V: x = g^r
V chooses random c
V → P: c
P → V: y = r + s*c
V verifies that x = g^y * t^-c

soundness
send c1 = r_1, get y1 = r + s*r_1
rewind and send c2 = r_2, get y2 = r + s*r_2
s = (y1 - y2) / (c1 - c2)

zero knowledge
pick random c, y
set x = g^y * t^-c

6.5 Or-Proof

prove knowledge of t1 = g^s1 or t2 = g^s2
done simultaneously, can forge knowledge of one (here s1 known)

public parameters
G with hard discrete log & generator g

knowledge of P
t1 = g^s1 but not necessarily s2 of t2 = g^s2
t1/t2 is known to V too (but without s1/s2)

protocol
P chooses random r1, r2, w
P → V: x1 = g^r1, x2 = g^r2 * t2^-w
V → P: c
P → V: c1 = c-w, c2 = w, y1 = r1 + s1*c1, y2 = r2
V verifies c = c1 + c2, x1 = g^y1 * t1^-c1, x2 = g^y2 * t2^-c2

soundness
send c1 = r_1, get y1 & y2
rewind and send c2 = r_2
then can calculate s

zero knowledge
choose random c1,c2,y1,y2 then calculate the rest

non-interactive scheme
use hash of all public values as challenge c

6.6 pederson proof

prove knowledge of C = g^x * h^r for random r, generators g h

public parameters
G with hard discrete log & generator g, h with unknown log

knowledge of P
C = g^x * h^r
C is known to V too (but without x,s)

protocol
P chooses r1, r2
P → V : t = g^r1 * h^r2
V → P : c
P → V : y1 = r1 + c*x, y2 = r2 + c*r
V verifies that t = g^y1 * h^y2 * C^-1

soundness
send c1 = r_1, get y1 & y2
rewind and send c2 = r_2, get more y1 & y2
then can calculate s

zero knowledge
pick random c, y1, y2; then calculate t

6.7 commitment scheme

X commits to Y to hidden value x
commit stage where X sends box to Y
reveal when X sends key to Y allowing it to open

properties
hiding (after commit, V learns nothing about x)
binding (after commit, X can't change x)
either computationally or perfect

naive ideas
encryption (but not binding)
hashing (but not hiding, because hash allowed to leak)

pederson
send G & generators g,h that h = g^a (setup of receiver)
send c = g^s * h^r for secret s and random r (commit)
send (s, r) (reveal), verifier checks c
perfectly hiding (any r' exists that c=g^s' * h^r')
perfectly binding (need a = DL(g, h) for x + ar = x' + ar')

confidential transactions
pederson is homomorphic (c1 = c2*c3 iff s1 = s2+s3 && r1 = r2+r3)
but overflows (s1 = 1, s2 = org(G), s3 = 1)
use OR proofs for all values in allowed range
for output O = g^s * h^r, proof DL knowledge of some m in g^s-m
make proof efficient by splitting s into bits
O = g^s_i*h^r_i for s_i bit, r_i such that sumof r_i = r
for each g^s_i*h^r_i use OR proof with c_i (for 0) or c_i*g^-2^i (for 1)