Cryptographic Protocols

58555 characters in 10348 words on 1577 lines

Florian Moser

August 27, 2020

1 mathematical foundations

1.1 group <G; ∗>

non empty set with binary association
(1) ∗ is associative like x∗(y∗z) = (x∗y)∗z
(2) e is the neutral element like x∗e = e∗x = x
(3) xˆ is the inverse x∗xˆ = xˆ ∗ x = e
for ∗ commutative (x∗y = y∗x) then abelian group
for ∗ denoted as +, inverse as -, e as 0 then additive
for ∗ denoted as ∗, inverse as ˆ-1, e as 1 then multiplicative

examples
<Z, +> are integers with addition
<R \ {0}; ∗> are reals with multiplication
<Z_n; op_n> are integers with operation over modulo

order
number of elements in G = |G|
ord(x) is least k such that x^k = e
ord(x) |G| (divides group order)
hence also xˆ|G| = e

cyclic
if finite group has generator <g>
such that G = {g^0, g^1, ...}

isomorphic
if bijection v: G → H exists such that
v(x ∗ y) = v(x) ∗ v(y)
groups are "the same", only element name differs

construct groups
<Z_m∗; ∗_m> as {x ∈ Z | 0 ≤ x < m, gcd(x,m) = 1}
group because gcd(x,m) implies (3); (1), (2) trivial
if m is prime, |G| = m-1 & all entries are generators
if m = pq for p,q primes, |G| = (p-1)(q-1)

1.2 modulo

congruency
x, y congruent mod (m) if same reminder
hence x-y mod m = 0

inverse
x∗y mod m = 1
hence x∗y, 1 congruent mod (m)

coprime
if gcd(x,y) = 1
can use EEA for a∗x + b∗y = 1

fermat's little theorem
x^p = x (for p prime group order)
hence also xˆ(p-1) = 1

1.3 extended euclidian algorithm (EEA)

calculates a∗x + b∗y = ggt(x,y)
works in two steps

find ggt(x,y)
write x = 1∗y + d_1
because x,y known, calculate d_1 trivially
write y = e_1∗d_1 + d_2
choose biggest e_1 such that e_1∗d_1 ≤ y
calculation of d_2 again trivial
continue d_1 = e_2∗d_2 + d_3, d_2 = e_3∗d_3 + d_4, ...
until reminder d_x = 0, hence d_(x-1)

reconstruct a,b
assuming d_5 is 0, hence d_4 is ggt(x,y)
write d_4 = d_2 - e_3∗d_3
replace d_3 with d_1 - e_2∗d_2
continue until no more d∗ on the left
you end up with a∗x + b∗y

properties
a∗x mod m = 1
hence a is inverse of x
any a, a' that fulfill a∗x mod m = 1 are congruent

1.4 RSA

generate public key k_{pub}, private key k_{priv}
messages x encrypted with k_{pub} can be decrypted with k_{priv}

pair generation
choose primes p,q
get m = pq, phi = (p-1)∗(q-1) = |Z_m|
choose e such that gcd(e,f) = 1
use EEA to get d∗e + k∗phi = 1
(n, e) as public key, (n, d) as private key

message transfer
encryption with x^e = c
decryption with c^d = xˆ(e∗d) = xˆ(-k∗phi+1) = x
because x^{phi} = 1 due to phi = ord(|Z_m|)

security
assumption that it is hard to factor m into p,q
only known to be hard in some models of computation
needs to be randomized for real applications

1.5 chinese remainder theorem (CRT)

given x mod m_1 = a_1, x mod m_2 = a_2, ...
there is unique x ≤ M = m_1 ∗ m_2 ∗ ...

unique solution
let M_i = M / m_i, hence gcd(Mi, mi) = 1
then M_i mod m_j = 0 for j!=i
for N_i inverse of M_i
solution x = $\sum a_i{*}M_i{*}N_i$ mod M

example
given equations in form x mod n_i = a_i
like x mod 3 = 2, x mod 4 = 5, x mod 7 = -3
calculate M = $\prod$(n_i)
like M = 3∗4∗7 = 84
find inverse Ni of M/n_i ∗ N_i mod n_i = 1
like 84/3 ∗ N_i mod 3 = 1 ⇒ N_i = 1
like N_1 = 1, N_2 = 1, N_3 = 3
calculate x = $\sum a_i{*}M_i{*}N_i$ mod M
like x = 2∗28∗1 + 5∗21∗1 + -3∗12∗3 mod 84 = 53

isomorphic groups
for m = pq <Z_m∗,∗_m> isomorphic to <Z_p∗ x Z_q∗, ∗_pq>
CRT allows to prove this isomorphism
for p=3,q=5, hence m=15
7 → (1,2) with (7 mod 3, 7 mod 5)
(1,2) → 7 with CRT on x mod 3 = 1, x mod 5 = 2

1.6 quadratic residue (QR)

numbers which are the result of a square
else called a quadratic non-residue (QnR)
a is QR iff ∃ r such that r^2 mod m = a
QnR ∗ QR = QnR; QR∗QR = QnR∗QnR = QR

find QRs
square each number from 1 up to (m+1)/2 for Z_m
calculate negative equivalent for > (m+1)/2
for root 5, n = 14 ⇒ 14 - 5 must also be a root

number of QRs
exactly (p-1)/2 for |Z_p∗| = p-1

hence each QR has two square roots r,r'
it holds that r mod p = r' mod p

legendre symbol
(a/p) (written as a fraction)
1 iff a QR over mod p
0 iff a divides p over mod p
-1 iff a QnR over mod p
satisfies multiplication rules

euler's criterion
(a/p) = a^((p-1)/2)
because (x^2)^((p-1)/2) = x^(p-1) = 1 (fermats little theorem)
it can be shown that all other numbers must equal -1
for p=5 (p-1)/2 = 2
$1 \rightarrow 1^2 = 1$, $2 \rightarrow 2^2 = 4$, $3 \rightarrow 3^2 = 9$, $4 \rightarrow 4^2 = 16$
verify that this indeed results in 1, -1, -1, 1

four square roots for Z_m*
for m = pq <Z_m*,*_m> isomorphic to <Z_p* x Z_q*, *_pq>
implies that r^2 = a ⇔ (R_p(r^2), R_q(r^2)) = (R_p(a), R_q(a))
hence four square roots for each QR in Z_m*
for p=3,q=5, (1,4) are QR in Z_3, Z_5; hence 4 is QR in Z_m

QR breaks RSA
assume A, which given a can calculate r^2 = a mod m
B chooses random r, asks A to solve a = r^2
A returns either r' = r or -r (randomly, bc a hides info about r)
if A returns r' == r, B has to abort (hence success p only 1/2)
else B calculates gcd(r+r', m) to get one of the primes
works bc (r-r')(r+r') = 0, each being a multiple of one of primes

1.7 two dimensional polynomials

of the form f_{00} + f_{01x} + f_{10y} + f_{11xy} + ...

fact 1
f(x, y_0) is one-dimensional polynomial of degree t
because $(f_{00} + f_{01}*y_0 + ...)x^0 + (f_{10} + f_{11}*y_0 + ...)x^1 + ...$

fact 2
(t)^2 values, t combinations uniquely define polynomial of degree d = t-1
choose d x values, and d y values
choose z_{ij} for each x/y combination
construct polynomial with lagrange-interpolation
show the polynomial in this form can only exist once

1.8 language L classification

using turing machine (TM) model
for input z, s(z) measures number of steps
t(n) = max{s(z) for all |z| ≤ n}
halting problem in neither of the presented languages

P
given candidate
membership in L can be efficiently decided
"there is poly-time TM deciding L"
for all z ∈ {0,1}* ∃ efficient A(z)
such that iff z ∈ L then A(z) = 1 else 0

NP
given candidate & proof ("witness")
membership in L can be efficiently accepted/rejected
"there is non-deterministic poly-time TM accepting L"
for polynomial p, poly-time computation o(candidate, proof) → accept/reject
such that iff z ∈ L then x exists |x| < p(|z|)
with o(z, x) = 1 (soundness)
else o(z, x) = 0 (correctness)

NP-hard
any NP language can be solved with this language
hence at least as hard as NP, potentially harder
for any L' ∈ NP, L' can be efficiently reduced to L

NP-complete
in NP and NP-hard
useful because NP-hard also include harder problems than NP

IP
efficiently verifiable interactive proof of membership exists
superset of NP bc interactive proofs more general concept than non-interactive

PSPACE
only polynomial space used (no constraint on time made)
"there is TM that uses only poly memory"
proven to include same problems as IP
IP $subset_{of}$ PSPACE bc can construct polynomial tree with all transcripts

1.9 algorithm classifications

efficient if running time grows at most polynomial with input size
unbounded if running time arbitrary
randomized/probabilistic if access to uniformly random bits
deterministic if no access to uniformly random bits

1.10 function classification

polynomial
it grows slower than some polynomial
starting at some n,
if ∃ c, n_0 ∈ n > n_0
such that f(n) ≤ n^c

negligible
decreases faster than inverse of every polynomial
if ∈ c, ∃ n_0, ∈ n > n_0
such that f(n) ≤ 1/(n^c)
other notions possible
but should stay negligible if repeated efficiently ofen

noticeable / non-negligible
grows faster than inverse of some polynomial
if ∃ c, n_0 ∈ n > n_0
such that f(n) ≥ 1/(n^c)

overwhelming
grows infinitely
1-f is negligible

calculation rules
polynomial ∗ polynomial = polynomial
polynomial ∗ negligible is negligible
polynomial ∗ noticeable might be overwhelming

1.11 decision problem

problem with answer accept/reject
like existence of hamiltonian cycle, isomorphims of graphs, ...

as formal language L
instances of problem as bitstring
bitstring z ∈ L iff decision true

2 proofs

2.1 primality proof

for small n, simply do table lookup
else decompose n-1 into p_1, p_2, ...
find a such that a^(n-1) mod n = 1
and a^(n-1)/p_i mod n != 1
then recursively proof primality for all p_i

2.2 proof system

statement & proof each are a string over finite alphabet
semantics define which statements are true
verification function calculates (statement, proof) → (accept or reject)

non-prime proof example
verification function checks if proof divides statement
statement = 12, proof = 3, output = accept

requirements
soundness (only true statements have proofs)
completeness (every true statement has a proof)
efficient verifiability (verification efficiently computable)

potential criteria
efficiency (for prover/verfier, messages & rounds)
generality (which type of statements can be proved)
leakage (what kind of information prover has to share with peggy)
type of security (information theoretic, based on RSA, based on DL, ...)

2.3 "what" proof types

proof of knowledge

proof some knowledge exist
like i know for sudoku X the solution Y

proof of statement
proof a statement, may follows from knowledge
like sudoku X has solution Y

2.4 "how" proof types

static proof
prover and verifier know statement s
prover sends proof p to verifier
verifier accepts/rejects (s,p) combination

interactive proof
proof string replaced by interaction with unbounded prover p
both prover & verifier are probabilistic
verifier/provers may deviate from the protocol
at the end of interaction, verifier accepts/rejects

2.5 interactive proof

proof for language L as pair of probabilistic algorithms (P,V)
if z $\in$ L then V accepts with at least p=3/4
if z $\notin$ L then V accepts with at most q=1/2
p and q can be arbitrary, but they must be $0 < q < p \leq 1$

language membership
transcript of deterministic P,V serves as NP witness
bc deterministic implies only single x witness exists
transcript of P,V with q=0 serves as NP witness
bc q=0 implies no wrong witness, p>q implies some witness

prover properties
can be deterministic (as powerful as indeterministic)
but deterministic can not be zero-knowledge
if poly-time required, then only "interactive argument"

verifier properties
randomized (else prover can be constructed trivially)
efficient (running time polynomial in |z|)

parallelization
can execute n rounds in parallel
only remains zero-knowledge if #rounds = O(log (n))

2.6 interactive proof applications

identification protocols
prover is able to identify itself to verifier
use hard problem (like hamiltonian graph, public key)
and prove knowledge about solution (hamiltonian cycle, private key)
must choose sufficiently hard but still efficient problem
NP not sufficient bc some instances may be easily solvable
first practical implementation by fiat-shamir

fiat-shamir heuristic
replace interactive proof with non interactive one
by calculating verifier input from hash function
then sending all messages at once to verifier
needs hash function to be random oracle (truly random results)
hence lives in the random oracle model (ROM)
but very useful in practice, because constructable from many problems

digital signatures
construct digital signature using the fiat-shamir heuristic
choose random instance z of NP problem with witness x
to sign message m, generate randoms t_1, t_2, ... (sufficiently many)
generate c_1, c_2, ... from hash(t_1, t_2, ..., m)
generate answers r_1, r_2, ...
signature then is (t_1, t_2, ...; c_1, c_2, ...; r_1, r_2, ...)
assumption that c_1, c_2, ... can not be sufficiently influenced to cheat

2.7 proof of knowledge

for string z, proof x
Q(z,x) = true iff x proofs knowledge of z

2.7.1 requirements

completeness
V accepts if P knows some x
such that Q(z,x) = true

soundness
there exists knowledge extractor K
which interacts with some P which V accepts noticeable
and then outputs valid secret x
(K can rewind P = restart with same randomness)

2.7.2 properties

2-extractability
if from two accepted rounds
with outputs (t,c,r) & (t,c',r') and c != c'
secret x can be efficiently computed
for 1/|C|^s (s=#rounds) negligible compared to input length |z|

three-move
protocol consists of exactly three moves
P $\rightarrow$ V some start value
V $\rightarrow$ P some challenge
P $\rightarrow$ V result calculated with challenge relative to start value

2.7.3 knowledge extractor example

for s-round 2-extractable 3-move protocol with negligible 1/|c|^s
(1) choose l uniformly at random
(2) generate two executions with same l
(3) iff V does not accepts both restart, else stop

soundness
use first round with c != c' to get x (2-extractability)
in first such round, t = t' bc prover randomness fixed

efficiency
for p probability V accepts
E[f(l)] = p for f(l) probability V accepts using random l
E[f(l)^2] $\geq$ (jensens inequality) E[f(l)]^2 = p^2 for two executions
because 1/|c|^s negligible, protocol execution equal can be ignored
hence runs in polynomial O(1/p^2) for p noticeable

2.7.4 witness-hiding & witness-independence

if proving zero-knowledge not possible
show that verifier can not impersonate prover

witness-hiding
no poly-time verification V after verification with P
can itself act as prover for another verifier V'

witness-independent
if for all z, all V'
distribution of transcript is identical for each witness

witness-independent $\Rightarrow$ witness-hiding
if hard to generate (z, w, w') for w != w'
but easy to generate (z, w)
then witness-independence implies witness-hiding

2.8 zero-knowledge

for protocol (P,V) resulting in transcript T
show simulator S exists which outputs indistinguishable transcript T'
for both proof of statements / knowledge

requirements
complete (true statements have proof)
soundness (only true statements have proof)
some classification of zero-knowledge

proof checks
completeness by inspection
soundness by showing that knowledge extractor exists / 2-extractable
zero-knowledge by showing c-simulatability & poly-space C
zero-knowledge classification depending on powers of verifier

distinguisher A
tries to differentiate T and T'
hence tries to decide "Y" on Y $\rightarrow$ y / "X" on X $\rightarrow$ x
advantage given by P_X[A(x) = "X"] - P_Y[A(y) = "Y"]

indistinguishable level
perfect (exact same distribution; like $P_X = P_Y$; advantage = 0)
statistical (difference is negligible)
computational (difference for poly-time algorithms negligible)

c-simulatability
(for t random input prover, c challenge verifier, r response prover)
if for any c, a triplet (t,c,r) can be generated
with same distribution as in a real execution of the protocol
hence it holds that P_{TR}|C = p_T * p_R|TC
for example given c, choose r uniformly, then generate t
"conditional distribution P_{TR}|C is efficiently samplable"

2.9 zero-knowledge classifications

proof checks
no challenge publishes the secret or any value leading to it
no dishonest verifier learns new information (then only HVZK)
size of challenge space must be polynomial

honest-verifier zero-knowledge (HVZK)
verifier used to generate T' must be honest
honest V chooses challenge independently of m
weaker than (perfect) ZK

zero-knowledge (ZK)
for all polytime V', input z, there is poly-time simulator S
such that transcript T and T' are indistinct
must also hold for dishonest prover
which picks challenge dependent on previously seen messages

black-box zero-knowledge (BB-ZK)
there is single poly-time simulator S for all polytime V', input z,
with S having rewind access to V
such that transcript T and T' are indistinct
hence stronger than (perfect) ZK bc only single simulator needed

construct ZK simulator for dishonest V
V chooses challenge c depending on previously seen messages
determines c_i it wants to do choose the next round
uses honest verifier V' to generate transcript
if transcript used c_i, then accept; hence repeat round
V is still efficient; expected runtime is 1/|c| (hence polynomial)

2.10 zero-knowledge proofs

proof that simulator with same distribution exists & is efficient

3-move distribution
for prover random t, verifier challenge c, prover reply r
both t and c chosen uniformly at random
then prover chooses r depending on t and c
hence expected distribution is $p_T * p_C * p_R$|TC
dishonest verifier may choose c based on t, hence p_C|T

(1) 3-move c-simulatable protocol ⇒ HVZK
honest verifier chooses c with p_C
then generate t & r with p_{TR}|C
due to c-simulatability same distribution as expected

(2) HVZK 3-move with poly |C| ⇒ BB-ZK
S generates triplet (t,c,r) with HVZK property
(hence distribution d_1 = p_T(t) ∗ p_R|CT ∗ 1/|C|)
then invokes verifier with t, getting c'
(we assume dishonest verifier, hence $d_2 = d_1 * p_C$|T)
if c' equals c, then output triplet; else restart
(probability e = 1/|C|, bc other terms are summed up)
hence distribution is d_2 / e = $p_T * p_C$|T ∗ p_R|TC
efficient bc polynomial runtime / success probability e

(3) sequence of BB-ZK is BB-ZK
build a simulator that uses the simulator of the respective sequence

s rounds of c-simulatable 3-move with poly |C| ⇒ ZK
poly |C| is needed bc else could not generate transcripts for all challenges easily
bc of (1), subprotocol HVZK
bc of (2), subprotocol BB-ZK
bc of (3), protocol BB-ZK
bc BB-ZK stricter than ZK, it follows that ZK

3-move HVZK with uniform challenge ⇒ c-simulatable
construct (P',V') using HVZK (P,V)
P sends t to P'
P' chooses c" and sends (t, c") to V'
V' chooses c' and sends it to P'
P' sends c = c' + c" to P
P answers with r to P', which forwards to V'
V' accepts/rejects (t, c'+c", r)
c-simulatable bc P' can simulate for any c
by choosing c" = c + c'

2.11 zero-knowledge discrete math examples

2.11.1 fiat-shamir

given m as RSA modulo, z ∈ Z_m∗
proof knowledge of x such that x^2 mod m = z

protocol
prover and verifier know z
prover knows x such that x^2 mod m = z
prover picks random k ∈ Z_m∗
prover sends t = k^2
verifier sends c ∈ {0,1}
prover sends r = k ∗ x^c
verifier checks that r^2 = t ∗ z^c

proof
c=0 works because $r^2 = k^2$ = t
c=1 works because $r^2 = k^2 * x^2$ = t ∗ z

extractability
get r_0 = k, r_1 = k ∗ x → extract x

2.11.2 guillou-quisquater

given m as RSA modulo, z ∈ Z_m∗, e
proof knowledge of e-th root of x, such that x^e = z

protocol
prover & verifier know z, e
prover knows x such that x^e = z
prover picks random k ∈ Z_m∗
prover sends t = k^e
verifier sends c ∈ {0,1, ..., e-1}
prover sends r = k ∗ x^c
verifier checks that r^e = t ∗ z^c

proof
works because $r^e = k^e$ ∗ xˆ(c∗e) = t ∗ z^c

extractability
get r_c = k ∗ x^c, r_d = k ∗ x^d
let ggt(c-d, e) = (c-d) ∗ a + e ∗ b = 1
then x = (r_c / r_d)ˆa ∗ z^b

2.11.3 schnorr

given cyclic group H, generator h, prime order q, z ∈ H
proof knowledge of discrete logarithm x of z

protocol
prover and verifier know z
prover knows x such that h^x mod m = z
prover picks random k ∈ Z_q∗
prover sends t = h^k
verifier sends c ∈ Z_q∗
prover sends r = k + x∗c
verifier checks that h^r = t ∗ z^c

proof
h^r = hˆ(k+x∗c) = t∗z^c

extractability
r_c = k + c∗x, r_d = k + d∗x
x = (r_c - r_d) / (c - d)

2.12 one-way group homomorphism (OWGH)

f : G → H such that [a ∗ b] = [a] x [b] (f written as [])

2.12.1 examples

exponential
G = H = <Z_m∗, ∗>, [a] = a^e
for example [a∗b] = (a∗b)ˆe = $a^e * b^e$ = [a] ∗ [b]

logarithmic
G = <Z_q, +>, H = <h>, [a] = h^a
for example [a+b] = hˆ(a+b) = $h^a * h^b$ = [a] ∗ [b]

2.12.2 pre-image proof of knowledge (OWGH PoK)

given groups G and H as one-way homomorphism f(x+y) = f(x) ∗ f(y)
proof knowledge of pre-image x ∈ G of z ∈ H

protocol
prover & verifier know z ∈ H
prover knows x ∈ G such that [x] = z
prover picks random k ∈ G
prover sends t = [k]
verifier sends c ∈ Z_+
prover sends r = k ∗ x^c
verifier checks that [r] = t ∗ z^c

proof
works because [r] = [k] ∗ [x]ˆc = t ∗ z^c

2.12.3 two-extractability of OWGH PoK

requirement
if ∃ l and u ∈ G
(1) for all c_1 != c_2, gcd(c_1 - c_2, l) = 1
(2) [u] = z^l

proof
let [r_1] = t ∗ z^{c1}, [r_2] = t ∗ z^{c2}
then x' = u^a + (r_1/r_2)ˆb
for gcd(c_1-c_2, l) = (c_1-c_2)∗a + l∗b = 1
works because u = z^l and r_1/r_2 = zˆ(c_1-c_2)

2.12.4 instantiation

choose appropriate homomorphism for []
prove that it holds by showing f(x ∗ y) = f(x) ∗ f(y)
define poly-bound C (for example Z_q)
argue that s (number of rounds) such that 1/|C|ˆs
choose l which is co-prime to all c_1, c_2
define u such that [u] = z^l (usually something with z)

schnorr
by definition G = Z_q, H = <h>, |H| = q, q is prime, [x] = h^x
choose l = q, hence u = 0
(1) gcd(c_1 - c_2, q) = 1 (bc q is prime)
(2) [0] = h^0 = 1 = z^q (bc of identity)

guillou-quisquater
by definition G = H = Z_m∗, [x] = x^e
choose l=e, hence u=z
(1) gcd(c_1 - c_2, e) = 1 (bc e is prime)
(2) [z] = z^e = z^l

2.13 zero-knowledge NP problem examples

ZK proof of NP-complete problem allows to do ZK for arbitrary NP

2.13.1 sudoku

proof that solution is known without revealing it

protocol
peggy places three cards per field with correct number
numbers visible if preset, hidden if part of solution
vic chooses for all column, row, cell one of the tree cards
(hence board empty at the end of the move)
gives the cards for each column, row, cell face-down to peggy
peggy shuffles each deck and returns them to vic
vic controls that each deck is valid

soundness
1/3 that proof succeeds although sudoku wrong
because have to pick the correct out of three cards

2.13.2 sudoku (using ZK for equality)

proof that solution is known without revealing it
uses type B commit protocol
uses ZK proof to show some blob of commitments are equal

protocol
peggy commits to every cell of the sudoku solution (1)
peggy commits to 1..n for every row/column/subgrid (2)
vic chooses challenge c = 0 or c = 1
if c == 0 then peggy opens (2) and preprinted values of (1)
vic checks that (2) consistent & (1) correct
if c == 1 then peggy use ZK to show (1) equals (2)

proof
completeness (bc peggy can answer both c if solution known)
soundness (approx. 1/2 that peggy succeeds if solution unknown)
only approx. 1/2 bc could cheat in ZK proof with negligible p
proof of knowledge bc of 2-extractability
get triplets (t, c, r) and (t, c', r'); one opens (2)
the other proves how opened values relate to (1) values
zero-knowledge bc of c-simulatability
commit as usual choosing random values for sudoku solution
for c=0 simply open (trivially correct)
for c=1 use simulator of ZK protocol
bc commitment is computational hiding
simulator output is computationally indisting

2.13.3 graph isomorphism

given two graphs G_0 and G_1
proof that G_0 and G_1 are isomorphic

protocol
prover & verifier know G_0, G_1
prover knows o such that G_1 = o ∗ G_0 ∗ o^{-1}
prover picks random permutation pi
prover sends T = pi ∗ G_0 ∗ pi^1
verifier sends c ∈ {0,1}
prover sends p = pi (for c=0)
or p = pi ∗ o^{-1} (for c=1)
verifier checks that T = p∗G_0∗p^{-1} (for c=0)
or T = pi∗G_1∗pi^{-1} (for c=1)

proof
c=0 works because of construction of T
c=1 T = pi ∗ o^{-1} ∗ G_1 ∗ o ∗ pi^{-1} = pi ∗ G_0 ∗ pi^{-1}

zero-knowledge
no; V now knows if G_0 and G_1 are isomorph

2.13.4 graph non-isomorphism (GNI)

given two graphs G_0 and G_1
proof that G_0 and G_1 are not isomorphic

protocol
prover & verifier know G_0, G_1
verifier picks random pi and b ∈ {0,1}
verifier sends T = pi ∗ G_b ∗ pi^{-1}
prover sends r=0 if T ˜ G_0
or r=1 if T ˜ G_1
verifier checks that r = b

zero-knowledge
only HV-ZK
else V chooses arbitrary graph K to learn if isomorph

three graph extension
with three graphs, verifier permutes each graph
then sends triple shifted by random factor
prover must be able to tell shift factor
only HK-ZK, bc else learn shift factor

2.13.5 graph coloring

proof of statement
given graph G
proof that a k-coloring exists

protocol
verifier picks random pi (bijection on vertices)
verifier applies pi to vertex color map f to get f'
verifier creates commitment for all f' & sends to prover
prover sends edge (i,j) to verifier
verifier opens commitments to C_i, C_j
prover accepts if committed color is different

zero-knowledge
yes, because c-simulatability given

2.13.6 hamiltonian cycle (hc)

given graph G_0
proof that G_0 has closed path visiting each node exactly once

adjacency matrix
matrix with 1 where directed graph has edge
select exactly one 1 in each row/column for hc

protocol
prover picks random permutation pi
verifier picks c ∈ {0,1}
if 0, prover uncovers whole adjacency matrix
verifier checks permutation is valid
if 1, prover uncovers 1 in each column/row (hence cycle)
verifier checks in each column/row exactly one 1
⇒ unclear how to simulate "uncover"

proof
check completeness by inspection
both c=0 and c=1 fulfillable by peggy
soundness when knowledge extractor exists
negligible, C witness, 2-extractable all given hence KE exists
2-extractable because with both responses can reconstruct answers
zero knowledge when simulator can output transcript
for c=0 choose random permutation
for c=1 choose H all 1's; open random HC
commitment must be type H (for c=1 case)

2.13.7 boolean circuit

lower reduction overhead than hc bc more cases representable
computes fulfilment of boolean circuit
let input bits flow through scrambled truth tables
after truth table add masking bit (0 or 1 mask)

scramble truth table
(1) on each wire, choose random bit and XOR input/output
hence if bit 1, invert truth table input column from wire
and truth table result / input bit leading to wire
(2) permute rows of truth table randomly

protocol
peggy permutes truth tables of whole circuit and commits
then computes result of permuted circuit
if vic chooses c=0 then peggy reveals circuit & random bits
hence peggy verifies the permutation was applied correctly
if vic chooses c=1 then peggy reveals masked input bits & hit rows in tables
then peggy verifies that hit rows indeed give asserted result

proof
completeness by inspection
soundness; for c=0 open blobs, then use c=1 to get valid row assignments
recover original input values bottom-up
zero-knowledge because c-simulatable
for c=0, scramble circuit, send all blobs & open all
for c=1, set to all 1 in output, then open random rows

2.13.8 boolean circuit 2

use zero knowledge proofs of equality instead of blinding bits

gate protocol
P randomly permutes function table and commit to elements
V chooses c = 0 or c = 1
if c = 0 then P opens all commitments of table
V checks if valid permutation
if c = 1 then P proofs ZK that blobs of matching row equal

protocol
P commits to all bits on wire
P uses gate protocol to show V that gates correct

3 protocol foundations

3.1 attackers

share state with all other attackers
passive attacker must follow protocol
active attacker may deviate
usually constraint protocol to max t attackers

3.2 security

information theoretical
no assumptions like RSA, one-way functions
proof scheme only breakable with negligible probability
usually assume authenticated, complete, synchronous network

cryptographic
assume primitives to be secure based on hardness assumption

3.3 oblivious transfer (OT)

property on channel
between sender → trusted party → receiver
all variants equivalent
all variants have string (instead of bit) variation (OST)

rabin-OT
sender sends s to TTP
TTP only forwards s in 50%, else bottom

1-2-OT
sender sends s_0, s_1 to TTP
receiver send i to TTP for i = {0,1}
TTP forwards selected s_i to receiver

1-k-OT
sender sends s_0, s_1, ... to TTP
receiver send i to TTP for i = {0,1, ...}
TTP forwards selected s_i to receiver

1-2-OT ⇒ 1-k-OT
do k rounds, with each round random r_i and value c_i
each round, send e_i, r_i over 1-2-OT
for $e_i = c_i$ XOR with all r_(i-1)
hence receiver pick randoms until at round i with c_i
now can decrypt r_i to c_i (because knows all r_(i-1))
but does not learn r_i (hence can not decrypt any later value e_(i+1))

1-2-OT ⇒ rabin-OT
choose i ∈ {0,1}
send b_i = b, b_(i-1) = 0 over 1-2-OT
send i
if receiver picked correct b_i, now learns b
else learns nothing (and knows it, bc i != receiver picked i)

rabin-OT ⇒ 1-2-OT
transfer k random bits r_i with rabin-OT (for k security parameter)
receiver learns expected 1/2 of these r_i
receiver chooses random c
receiver forms T_c = {index where received}, T_(c-1) = {index where not received)
receiver sends T_0 and T_1 to sender
sender XOR r_i for all in T_0 (=t_0) and same T_1 (=t_1)
sender sends $e_0 = t_0$ XOR b_0, $e_1 = t_1$ XOR b_1
now receiver can decrypt e_c with XOR t_c

guarantees
recipient learns only one of the strings
sender does not know which one

1-2-OST with RSA/AES
sender has secrets s_0, s_1; shares one with receiver
generate two pairs of RSA (n, e, d) for n similar
sender sends (n_0, e_0) and (n_1, e_1) to receiver
receiver chooses random r
receiver sends back u = r^(e_b) for some b = {0,1}
sender calculates k = u^(d) for both d
sender calculates y = AES_k(s) for both s
sender sends y_0, y_1 to receiver
receiver gets s_b = AES_decrypt_r(y_b)
works because r^(e_b)^d = r for correct d
⇒ can be generalized to 1-k-OST

3.4 secret sharing scheme

dealer D shares secret s among parties P
qualified subset of P reconstruct s (without needing D)
access structure L defines who is able to reconstruct

definition
for protocol (share, reconstruct)
with parties P, access structure L
(1) after share, there is a unique value s'
where s' = s of dealer if dealer honest
(2) after reconstruct(M), iff M $subset_{of}$ L, M knows s'
(3) after share, all M' not_subset_of L do not know s'
(1),(2) for correctness, (3) for privacy

for L = {P}
n parties, L = {P} (hence all parties required for secret)
(share) send random xi to Pi such that $\sum$ xi = s
(reconstruct) all parties send each other xi

for L = arbitrary
n parties, L = arbitrary
(share) for each M ∈ L
send random xi to Pi such that $\sum$ xi = s
(reconstruct) parties ∈ M send each other specific xi
(hence parties receive multiple xi if in multiple M)

linear sharing scheme
if secret s and randoms r_1, ..., r_m
can be used to calculate player secrets s_1, ..., s_n
define as matrix multiplication with A as m∗n "decompose-matrix"
[s_1, s_2, ...] = [A_{10}, A_{11}, ...; ...; ..., A_{nm}] ∗ [s, r_1, ..., r_m]

3.5 shamir's secret sharing scheme

n parties, L = {any M for |M| ≥ k}
hence k parties needed for reconstruction

polynomial construction
construct polynomial f of degree d = k-1 with secret s at f(0)
hence of the form f(x) = s + a_1 ∗ x + a_2 ∗ x^2 + ... + a_(k-1) ∗ x^(k-1)
for a_i picked randomly
matrix A looks like [...; 1, a_i, a_i^2; ...] ("van der monde matrix")

lagrange
for y_i(x) = $\prod$((x - a_i)/(a_i - a_j)), s_i = f(a_i)
f(x) = $\sum$(y_i(x) $*$ s_i)
hence allows to calculate value at x with (a_i, s_i) tuples
for polynomial of degree k, k tuples needed

protocol
(share) choose f with degree d with f(0) = secret
choose n points a_i on f such that a_i != 0
send (a_i, s_i) for s_i = f(a_i) to each party P_i
(reconstruct) send (a_i, s_i) to P
P reconstructs s with lagrange interpolation on x = 0
note that points a_i can be public; only s_i must be party private

analysis if definition holds
(1) bc f provides single secret at f(0)
(2) bc with k shares, lagrange can output value
(3) bc with < k shares, any secret still compatible
⇒ degree must be d = k-1 for (3) to hold

violate privacy with k-1
construct d-1 polynomial g out of k-1 shares
now know that g(0) is not real value
bc else g would be equal to real polynomial

4 broadcast & consensus

4.1 known thresholds

for crypto, t < n (but consensus undefined for t ≥ n/2)
for theoretic, t < n/3

4.2 broadcast

single sender sends message to many receivers
input x; output y_1, y_2, ...

definition
(of course only need to hold for honest players)
consistency (y$*$ all equal)
validity (if sender honest ⇒ y$*$ = x)
termination (y eventually received)

behaviour
players output same y$*$
sender honest ⇒ players output y$*$=x

4.3 consensus

many players agree on single value of majority
input x_1, x_2 ...; output y_1, y_2, ...
undefined for t ≥ n/2 (because majority unclear)
"pre-agreement" if all honest provide same input

definition
(of course only need to hold for honest players)
consistency (y$*$ all equal)
persistency (if all honest same input x ⇒ y$*$ = x)
termination (y eventually received)

behaviour
players output same y$*$
pre-agreement ⇒ players output y$*$=x

4.4 consensus vs broadcast

given consensus, construct broadcast
P_1 sends x to all Pj which receive xj
(y_1, y_2, ...) = consensus(x_1, x_2, ...)
Pj output yj

given broadcast, construct consensus
Pi broadcast xi
yj = majority of received xi
Pj output yj

construction
weak → graded → king → "normal" consensus
then broadcast is archived

4.5 weak consensus

players output (y_i or bottom) such that all y_i are equal
input x_1, x_2 ...; output y_1, y_2, ...

properties
weak consistency (all y$*$ equal or bottom)
persistency (if all honest same input x ⇒ y$*$ = x)
termination (y eventually received)

protocol
send xi to every Pj
if (#zeros ≥ n-t) then yj=0
else if (#ones ≥ n-t) then yj=1
else yj=bottom
return yj

proof weak consistency
if Pi outputs 0, it received ≥ n-t zeros
hence Pj received ≥ n-2t zeros (bc at most t malicious)
hence Pj received ≤ 2t < n-t ones

4.6 graded consensus

input x_1,x_2,...; output (y_1, g_1), (y_2, g_2),...
for g grade, "how secure I am with this choice"

properties
graded consistency (if some p has (y, g=1) ⇒ y$*$ = (y, $*$))
graded persistency (if all honest same input x ⇒ y$*$ = (x, 1))
termination (y eventually received)

protocol
(z_1, z_2, ...) = weak_consensus(x_1, x_2, ...)
Pi sends zi to all Pj
if (#zeros ≥ #ones) then y = 0 else y = 1
if (#y ≥ n-t) then g = 1
return (y, g)

proof graded consistency
assume party outputs (0, 1) (hence #zeros ≥ n-t)
then for others (#zeros ≥ n-2t) > (#ones ≤ t)
because weak consensus implies no honest party published ones

4.7 king consensus

take own value if sure, else take kings value
input x_1, x_2, ...; output y_1, y_2, ...

properties
king consensus (if king correct ⇒ y$*$ = y)
persistency (if all honest same input x ⇒ y$*$ = x)
termination (y eventually received)

protocol
((z_1,g_1), (z_2, g_2), ...) = graded_consensus(x_1, x_2, ...)
Pk (king) sends (zk, gk) to all Pj
for all Pj if (gj = 1) then y = zj else y = zk (kings value)
return y

proof king consistency
assume king is honest
for g_j = 0, then take kings value
for g_j = 1, then all honest have same value (bc graded_consensus)
hence in both cases, king's value is taken

4.8 consensus

do kings consensus t+1 times; giving output as input again
first honest king archives consensus (due to king consensus)
consensus is kept until the end (due to persistency)

protocol
(repeat t+1 times for t+1 different kings)
(x_1, x_2, ...) = king_consensus(x_1, x_2, ...)

4.9 impossibility for P=3, t=1

if honest players both input 1, must decide 1
if honest players both input 0, must decide 0
if honest players have different input, must decide on same
honest player can not differentiate situations
hence can not fulfil consistency / persistency at same time

formal proof
assume consensus protocol for n=3, t=1
assume three programs pi_i (x_i ⇒ y_i), used by player P_i
each program has two channels for input/output to other player
attacker corrupts some player P_i, starts programs & connects channels
makes different cases (with different required output) look like the same
c_1 = (P_1 compromised, x_2 = 1, x_3 = 1) → output 1

$c_2 = (x_1 = 0, P_2$ compromised, $x_3 = 0) \rightarrow$ output 0
$c_3 = (x_1 = 0, x_2 = 1, P_3$ compromised$) \rightarrow$ output $y_1 = y_2$
for c_1, starts pi_1, pi_3 with both input 0
for c_2, starts pi_2, pi_3 with both input 1
for c_3, starts pi_3, pi_3 with input 0 and 1 respectively
in each case, connects programs such that same layout produced

5 commitment schemes

peggy P commits to value x towards vic V
P can open x at some point in the future

5.1 definition

P inputs x in COMMIT
V outputs x' in OPEN

properties
binding (after commit, x is fixed)
hiding (with commit, V does not learn x)
"non-exploitable" information; total security not required

correctness
if vic honest, x' ∈ {x, bottom}
if both honest, x' = x

5.2 trivial implementations

hash function h
send h(x) to V (COMMIT)
send x to V (OPEN)
but same value can only be committed to once
needs random oracle heuristic for h

hash function with random r
send h(r || x) to V (COMMIT)
send (r, x) to V (OPEN)
but security depends on hash function

5.3 commitment schemes (non-interactive)

function C(x, r) → b
P sends b = C(x,r) (the blob) to V (COMMIT)
P sends (x,r); V checks that b = C(x,r) (OPEN)

perfectly binding (Type B)
unbounded peggy cannot open x' != x
(at least) computational hiding
interactive proof bc result unchangeable (but hiding breakable)

perfectly hiding (Type H)
unbounded vic cannot obtain x
(at least) computational binding
interactive argument bc could cheat

simultaneous Type B / Type H not archivable
for Type B, only single message producable for same commit
formally, C(x, r) intersect C(x', r) = empty
for Type H, two different messages must be producable
formally, C(x, r) = c(x', r)
hence both at the same time impossible

combine schemes
like C_B(C_H(x,r_1), r_2) (1) or (C_B(x,r_1), C_H(x, r_2)) (2)
perfectly hiding as soon as chained (like (1))
else computational (like due to C_B in (2))
perfectly binding as soon as parallel (like (2))
else computational (like due to C_H in (1))
for computational binding (2), proof by contradiction
assume efficient x!=x' for C_B
hence C_H(x, r_1) = C_H(x', r_2), breaks computational hinding of C_H

5.4 discrete log scheme (type B)

cyclic group H of prime order q = |H|
generators g of H

COMMIT
x ∈ Z_p (value)
send C(x) = (g^x)

OPEN
send x

perfectly binding
g^x only single result

computational hiding
come up with x for g^x
if possible value room is small, trivial to break
like for x = age; simply bruteforce from 0 - 100

5.5 pedersen commitment scheme (type H)

cyclic group H of prime order q = |H|
generators g and h of H
with unknown discrete logarithm of g to h (DL_g(h))

COMMIT
x ∈ Z_p (value)
r ∈ Z_p (random)
send C(x,r) = $g^x * h^r$

OPEN
send x, r

perfectly hiding
uniformly random r ⇒ uniformly random h^r
hence h^r, g^r is also uniformly random

computational binding
come up with r', x' such that $h^r * g^x$ equals old value
efficiently possible with known DL_g(h) ("trapdoor")

5.6 elgamal commitment scheme (type B)

cyclic group H of prime order q = |H|
generators g and h of H
with unknown discrete logarithm of g to h (DL_g(h))

COMMIT
x ∈ Z_p (value)
r ∈ Z_p (random)
send C(x,r) = (g^r, $g^x * h^r$)
for g in second part, could use third generator with unknown DL

OPEN
send x, r

perfectly binding
g^r defines r, hence h^r also unique
h^r defines g^x, hence x also unique

computational hiding
come up with r for g^r
this defines h^r, which gives g^x
come up with x for g^x
efficiently possible with known DL_g(x) ("trapdoor")

5.7 homomorphic

C(x,r) op C(x',r') = C(x op x', r op r')
examples are DL, pedersons, ElGamal

implication
if commitment to a and to b known
then results in commitment to c = a op b

6 multi-party commitments

P_i commits to value x towards all parties
either all parties accept or reject (same) value

6.1 properties

consistency (D honest ⇒ no rejections, some rejections ⇒ all rejections)
privacy (hiding; COMMIT does not leak to attacker)
uniqueness (binding; OPEN only with single value)

6.2 construction

given non-interactive commitment scheme C
COMMIT with b=C(x,r) & broadcast b
OPEN with broadcast (x,r); accept x if b = C(x,r)

6.3 distributed commit scheme with shamirs secret sharing

use bivariate polynomial f(x,y) with single degree t with secret at f(0, 0)
each player gets (secret) value at some (public) point a_i

commit

D send each party projection on x coordinate and y coordinate
hence sends P_i (h_i(x) = f(x, a_i), k_i(y) = f(a_i, y))
[consistency checks]
each P_i compares with each P_j for consistency
P_i sends h_i(a_j) to P_j, which checks that equals to k_j(a_i)
if inconsistent, accuse dealer and broadcast (a_i, a_j)
dealer must broadcast f(a_i, a_j)
[accusations]
P_i checks if f(a_i, a_j) == k_i(a_j), P_j checks if f(a_i, a_j) == h_j(a_i))
if inconsistent, accuse dealer and broadcast
dealer must broadcast h_i and k_i of accusing player
players compare h_i and k_i with their own h_l, k_l
like h_i(a_l) = k_l(a_i) and k_i(a_l) = h_l(a_i)
if inconsistent, accuse dealer and broadcast
dealer must broadcasts h_l, k_l of all accusing players
accusing players use broadcasted h_l, k_l in future calculations
[determine commit-share]
if more than t accusations in step before, then disqualify dealer
else all without accusation calculate s_i = h_i(0), others s_i = k_i(0)
dealer outputs g(x) = f(x, 0)
(note this is the polynomial opened at the end)

open
dealer broadcasts coefficients of polynomial
players check if their share lays on the polynomial
accepted if at most t players accuse D
if D publishes f' != f with same degree t but different 0 value
hence at most t places equal than f
therefore at most n - t (bc equal) - t (bc dishonest) accept, still > t

requirements
privacy fulfilled; no information published attacker not already has
only up to t published, hence still any secret compatible
correctness trivial if dealer is honest
else after accusations, polynomials pairwise compatible for honest
assume f'(x,y) of degree d is determined by t+1 honest players
then also compatible with other honest player's h_i, k_i (bc of pairwise)

improvement to only 2 accusations
then dealer disqualified or no further accus. by honest parties
bc not disqualified in first round, t+1 honest players consistent
(hence polynomial of degree d already defined)
in round 2 dealer has to publish h & k of some party
must lie on unique polynomial else all honest accuse (and disqualify)

6.4 commitment transfer protocol (CTP)

transfer commitment to other party
for seen schemes, simply send the secret values

example shamir's sharing
send polynomial g to new party
all parties send secrets to new party
if $\geq$ n-t secrets lie on g, accept
else accuse using broadcast

6.5 commitment sharing protocol (CSP)

dealer is committed to some value s
want that all players are committed to some share of s

protocol
dealer chooses random coefficients and commits to them
hence all P_i now know all commitments c_i
players use homomorphy to check if $\prod c_i$ = c of s
dealer uses CTP to transfer corresponding share to Pi

6.6 commitment multiplication proof (CMP)

dealer is committed to some value a,b
want that dealer is committed to value c = a∗b

protocol
D calculates c = a∗b, commits to c
D executes CSP for a,b with degree t (using f(x) with f(0) = a, g(y) with g(0) = b)
hence P_i are committed to a_i and b_i
D executes CSP for c with degree 2t (using h(x) = f(x) ∗ g(x))
hence P_i are committed to c_i
all P_i check that $c_i = a_i * b_i$
if no player is able to open commitments to a_i, b_i and c_i
such that $a_i * b_i \neq c_i$
then protocol successful

7 multiparty computation (MPC)

enable n mutually distrusting parties
to compute function on their input
without revealing information not revealed by output

7.1 examples

real world
statistics (first intercourse, tax evading, ...)
elections / votes / auctions
millionairs problems
loans (customer at different banks but same guarantees)
zero-knowledge (ZK) proofs (isomorphic graphs)
stock market (buy/sell at some price, TTP is exchange)

secure function evaluation
send x_i to TTP
TPP computes some f(x_1, x_2, ...) = (y_1, y_2, ...)
receive back securely computed y_i

limitations
even passive attackers share state
if sharing state is advantage, then without TPP impossible
for example poker as attacker state influences victims state

7.2 MPC protocol

specification with trusted third party (TTP)
translated to protocol without TTP but same security

secure
if the adversary cannot achieve anything
that he could not achieve in the specification

simulator
to prove that "not worse" than specification
create a simulator for each adversary
that if would execute same steps under specification
would get equal outcome (same result, nothing new learned)

properties
defined relative to specification
privacy (nothing additional learned)
correctness (output cannot be falsified by dishonest)
fairness (cannot abort with advantage)
means output learned at the same time by all
hence can not abort as soon as result known to prevent others from knowing it too
robustness (protocol abortion impossible)

known thresholds
for crypto, t < n for passive, t < n/2 for active
for theoretic, t < n/2 for passive, t < n/3 for active
or assuming broadcast, t < n/2 for active

7.3 compute sum of inputs

specification
P_i have input x_i
sent to x_i to TTP
TTP sums and sends y to P_i

guarantees as given by specification
honest parties never learn other's values
honest parties finish with same sum
attackers may choose input / output value

ring protocol
initiator sends random r to first party
each adds own value and forwards to next party in order
initiator receives result, subtracts r and broadcasts result
$\geq$ 2 passive attacker uncover (for example, of party in between)

distributed sum
each party creates n x_{ij} random parts of own value with sum x_i
sends x_{ij} to Pj
Pj sums up received $x_{ji} = y_j$
Pj sends y_j to all Pi as a_j
Pi sums up all a_j for result
t < n passive attackers supported
single active attacker can decide result (by sending last)

7.4 MPC from OT

alice, bob with own secrets a, b

public fixed function F AxB→C
want to know output without other parties value

protocol
alice sends [f(a,b_1), f(a,b_2), ...] via 1-k-OST
bob chooses b'th value
bob sends value to a

guarantees
bob/alice learns result
bob/alice never sees each other's input

invertable function
with invertable function, bob learns a's value from result
but behaviour already possible in specification
hence does not invalidate protocol

generalization to 3 parties
send to party 2 table of evaluations
hence entry b_1 contains f(a, b_1, c_1), f(a, b_1, c_2), ...
but insecure against passive attacker party 2
for example if f(x,y,z) = 1 iff x=y=z
need to use one-time pad to encrypt table entries for party 2
party 1 sends keys directly (and only) to party 1

7.5 t < n/2 as an upper bound (passive)

two parties A, B running probabilistic program pi
calculate AND over their two two bits

game
A, B have input bit a,b and random r_a, r_b (for probabilistic)
execute protocol to get transcript T depending on input (naturally)

analysis
if b = 0, then transcript must not contain info about a
else contradicts privacy (nothing additional must be learned)
hence output must be distributed identically as r_b
if b = 1, then transcript must be influenced by a's value
else contradicts correctness (result calculated from input)
hence output distributed differently for a=0 and a=1

how to get B's input from T
A counts how often exact transcript was produced
(possible bc if A sends same message, B must respond same)
for input (a = 0, any r_a) and (a = 1, any r_a)
if for both a=0 and a=1 only negligible difference in occurrence
then b was 0 (bc no info about a in transcript)
else b was 1 (bc transcript dependent of a's input)

generalization to n
assume protocol exist supporting t > n/2 for n > 2
let task be "calculate AND", two parties with input bit, others bottom
then form two player groups M_1, M_2
let A simulate all players of M_1, and B all of M_2
contradicts that such a protocol can exist

active MPC attack
assume protocol with shortest messages sent & alice's turn
alice can cheat by simply not sending last message
alice must still learn result (bc no more input)
but bob does not (else not shortest protocol)
generalize to protocol with varying rounds
by arguing with non-negligible success probability
cryptographic implementations can not fix attack

binary function computations
for even-number of 1's (like XOR) MPC possible
use trivial procotol (simply exchange input)
bc output of function reveals inputs anyways (specification fulfilled)
for uneven-number of 1's can reduce to AND case
hence impossible

7.6 MPC computation in steps

compute in steps (each one being add/mult or XOR/AND in binary)
every intermediate result shared under n players with secret sharing

phases
input (input is shared between the players)
computation (computation takes place, preserving invariant)
output (output-receiving player receives shares of all others)

abstraction to players
users send input to players
not necessarily #players = #users

example distributed sum 2
users split input and send x_{ij} to players
players sum and send y_i to users

computations
addition / multiplication due to linearity easy
random with each party sending random r_i
inversion by multiplying x^(p-2), using square-and-multiply (log(p))
fast inversion by letting parties invert y = x∗r; then $x^{-1} = y^{-1} *$ r
x iff c=0, else y ⇒ c_{inv} = c^(p-1), then z = (1-c_{inv}) ∗ x + c_{inv} ∗ y
zero knowledge proofs with parties agreeing on single challenge

7.7 MPC computation with additions/multiplications

first share input
then execute additions / multiplications
at the end reconstruct output
(singular notation means single party, plural many)

share input by P_i
let P_i have s
P_i selects random r_1, r_2, ...
P_i computes (s_1, s_2, ...) = A ∗ (s, r_1, r_2, ...)
P_i sends s_i to every P_j

addition
let a,b be shared as a_1, a_2, ... and b_1, b_2, ...
P_i compute $c_i = a_i + b_i$
privacy (bc no communication)
correctness (due to linearity of operator like addition)

linear function
same as addition; simply replace addition by function

multiplication (naive approach)
let a,b be shared as a_1, a_2, ... and b_1, b_2, ...
P_i compute $d_i = a_i * b_i$
degree of polynom now at 2t (2t ≤ n, hence reconstruction still possible)
but repeated multiplication not possible
but polynom is reducable (contradicts privacy)

multiplication
let a,b be shared as a_1, a_2, ... and b_1, b_2, ...
P_i compute $d_i = a_i * b_i$
P_i share d_i with all other players using shamir sharing
hence receive (d_{1j}, d_{2j}, ...) from others
P_j calculate $c_j = \sum w_i * d_{ij}$
for w is weight according to lagrange
hence $w_i = \prod a_k$ / (a_k - a_i) for all k != j
new shares c_j on random poly with degree ≤ t

reconstruct output P_j
let a be shared as a_1, a_2, ...
P_i send a_i to P_j
P_j computes a = L(a_1, a_2, ...)

correctness
verify for each operation privacy & correctness
argue with no communication or underlying used protocol
share OK, bc new randoms are generated
addition OK, bc no communication
multiplication privacy bc either no communication / already known
multiplication correctness bc polynomial has degree 2t < n-t
reconstruction OK, bc only reconstructors get any message

7.8 MPC computation with additions/multiplications corruption

for t < n/2 attackers
when broadcast is needed, only t < n/3 attackers
must use broadcast for accusations
(0) publish secret information (passive corruption)
(1) + send additional messages
(2) + withhold messages
(3) + send wrong messages

7.8.1 protect against (0)

OK, because any t players can not reconstruct output
same than previous assumption as passive attackers already share state

7.8.2 protect against (1)

honest parties ignore messages not specified in the protocol
corrupted parties knew messages anyways (bc of shared state)
hence no advantage

7.8.3 protect against (2)

[share input] Pi does not send sj to all Pj
Pj broadcasts accusation
upon which Pi broadcasts its secret
if Pj corrupted, will receive broadcast
OK bc attacker knew secret already through corrupted party
if Pi corrupted & withholds, can choose any value (like 0)
OK bc others do the same

[multiplication] Pi does not share di
Pj broadcasts accusation
upon which Pi broadcasts its secret
if Pj corrupted, will receive broadcast
OK bc attacker knew secret already through corrupted party
if Pi corrupted, problem bc share is needed for algorithm
(1) repeat without dealer (but slow)
(2) reconstruct dealer share with any t+1 honest parties
(3) (2) + eliminate dealer; broadcast messages to eliminated players
(3) eliminate dealer, reshuffle results with t-1 assumed attackers

[reconstruct output] Pi receives not enough shares
up to t will not send, but can reconstruct polynomials
because n-t received is enough

7.8.4 protect against (3)

detect wrong messages and treat them like missing values
for detection, force participants to commit values

commitment types
on value send, include commitment in transfer
on value broadcast, sender must open commitment to all players
on value computation, commit to result & prove correctness (e.g. with ZK proofs)
in practice, always broadcast commitment
else could simply send different commitments to different players

required commit properties
(1) commit to single value
(2) open commitment to some other player
(3) transfer commitment to other player
(4) combine two commitments into one, adding values
(5) combine two commitments into one, multiplying values

analysis
(1) and (2) part of any commitment scheme
(4) need commitment scheme which is homomorph
(3) use commitment transfer protocol (CTP)
usually just send commitment & trapdoor to receiver
(5) use commitment multiplication protocol (CMP)
"need homomorph commitment scheme with CTP & CMP and broadcast"

protocol construction
[sharing input] make all players commit, use CTP to transfer
[adding / applying linear function] use homomorphic property
[multiplying] use CMP for mult, homomorphic for (linear) lagrange computation
[reconstruct output] players open commitment to receiver

security level
variate commitment scheme according to security level
for computational, use cryptographic like Pedersen or ElGamal
for theoretical, use distributed commitment schemes

computational implementation
as homomorph commitment use discrete log, pederson, elgamal
CTP trivial for all three; simply transfer secrets
CMP using ZK proofs (constructed depending on scheme)
for n verifiers, execute ZK proofs n times
or let verifiers construct single challenge to simulate single verifier
let each verifier choose & commit to c_i, use CTP, then c = L(c_i)

theoretical implementation
use distributed commit protocol
holds for t < n/3, for assumed broadcast even t < n/2
same holds for MPC (by construction)

8 blockchain

8.1 bank

bank keeps track of all balances
users request transactions (source, target, amount)
bank executes valid transactions

transactions on ledger
ledger where anybody can append/read entries
but no one can modify/delete entries
initialized with initial balances
balances derived from initials & transactions
but no privacy (only pseudo anonymity)
consistency because all users agree on state

sign transactions
account number is public key
transactions must be signed
but can still replay, link transactions

add nonces
add nonce field to transaction
use nonce=counter to check uniqueness easy
correctness because signature & no replay

transaction validity check
amount nonnegative & available at sender
signature signed under public key sender
target account valid
bit-string not executed before

escrow account
require k signatures for account by n parties
A transfers fund to shared account (k=2, n=3)
B does real-world transaction
when A & B agree, both sign and transfer fund to B
if not, judge J uses its signature to break to for A or B

8.2 trusted third party (TTP)

each user sends new entries to TTP
TTP appends to ledger and distributes new ledger

introduce blocks
TPP collects multiple valid transactions in blocks
adds hash of previous valid block

block valid check
syntactically correct
hash of previous valid block
transactions included all valid

distribute trust with MPC
parties simulate the ledger together
but only fixed & small number of participants possible
but needs synchronous communication

distribute trust without MPC
users send new entries to all parties
parties maintain local pool of unposted transactions
(hence liveness with at least one honest party)
king is chosen which forms & broadcasts new block
parties store received block & forward to users
(hence consistency for parties for t<n/3)
user chooses block received most often
(hence consistency for users for majority honest)
must only accept valid blocks

decouple users
improve performance protocol with less messages
user sends new entries only to some parties
(for liveness at least one honest party needed)
user requests blocks from some parties
(for consistency, majority must be honest)

8.3 permissionless

anyone is able to participate in network (eg create blocks)
alternative called "permissioned"

setting
multicast channel realised as peer-to-peer network
honest messages received by everyone

spam problem
assuming identity / sending messages cheap, hence attack easy
receiver-filtering has high cost & false positives
hence use proof-of-work of sender

partial hash inversion proof of work
given msg, find H(msg || nonce) for any nonce
such that results starts with D zeros
needs an average of 2^D guesses (hard)
verification is one hash query (easy)

8.4 block lottery

players hold copy of ledger locally
try to guess correct nonce to publish next block

block
consist of message, nonce, previous block hash
players try to guess nonce to append new block

validity
correctly formatted
only valid transactions contained
hash of last valid block contained
hash value with at least D 0s

signatures
to avoid players having to hold ledger locally
players request at most t+1 parties (bc could withhold)
then only need to verify t+1 signatures (bc more than attackers)

8.5 bitcoin protocol

users multicast their entries, parties collect
parties try to extend longest chain with fitting nonce
winner party multicasts block with new entries
parties/users append local ledges with valid new blocks

observations
difficulty D and total available power determine production
if multiple winners then block tree is created
these branches prevent instant confirmation

setting difficulty
low allows fast reproduction (but many branches)
high has few reproduction (but slow)
winner should know previous block
measure reproduction rate & adjust D automatically
bitcoin targets 10min per block

entry confirmation
probabilistic consensus; agree when rollbacks unlikely
hence confirm if entry k blocks deep on longest chain

problems
specialized hardware/centralization skews the lottery
scaling does not improve performance
consumes more energy than switzerland

8.6 reorganisation attack

active adversary with limited computation power
adversary published block b with entry e
creates second branch, starting before entry e
adversary waits until receives compensation for entry
then publishes longer second branch
hence entry e is no longer part of chain

observations
attacker needs to build chain faster than rest of network
hence needs majority of computing power
make attack harder by choosing large confirmation k

8.7 proof of stake

use money as limited resource
select blocks proportional to wealth
prevents sibyl attacks (more identities not helpful)

lottery approach
lottery winner proposes block
lottery tickets proportional to stake

byzantine fault tolerance
king replaced by winner of BFT
broadcast simulated by committee
committee chosen by lottery

randomness source
need to generate randomness to draw from lottery
use some form of MPC
but expensive & unclear how to choose participating parties
use verifiable random functions (VRF)
compute VRF($king_{me}$, random) and publish result with proof
others can verify proof with public key system

initial
needs initial stake distribution
or start with proof of work, then switch later

8.8 actual ledgers protocols

bitcoin with PoW, $t < n/2$
ethereum like bitcoin + smart contracts
cardano PoS bitcoin-like, $t < n/2$
algorand PoS BFT, $t < n/3$; no branching possible

8.9 construct bank

minting
miners need incentive to invest energy
creator of block gets block reward
could restrict total number of possible coins
bitcoins halves over time, fixes at 21 million

transaction fees
users could overwhelm the system with transactions
each transaction pays fee from sender to block creator
higher fee transactions processed first
in bitcoin, fees will replace minting

smart contracts
conditional transactions or other contracts
allows for investments, insurance, games
expressed as code, hence unambiguous execution
chain contract execution states like transactions
but publicly visible, bugs unfixable

mining pools
together with other parties work on same nonce
trusted party then commits solution to network
distributes reward proportionate to participating parties
estimate proportion by letting parties solve smaller challenges

8.10 privacy

ledger is public; transactions reveal amounts, accounts

hide balances
can encrypt balances/transactions with public key scheme
sent_amount under pk receiver
updated_amount under encryption sender
zero-knowledge proof checks validity

anonymous transactions
hide sender & receiver of transactions
needs many zero-knowledge proofs

real world
privacy level often unclear
monero had bug allowing to link transactions
zcash has opt-in & small anonymity set