Outsourcing computations and security

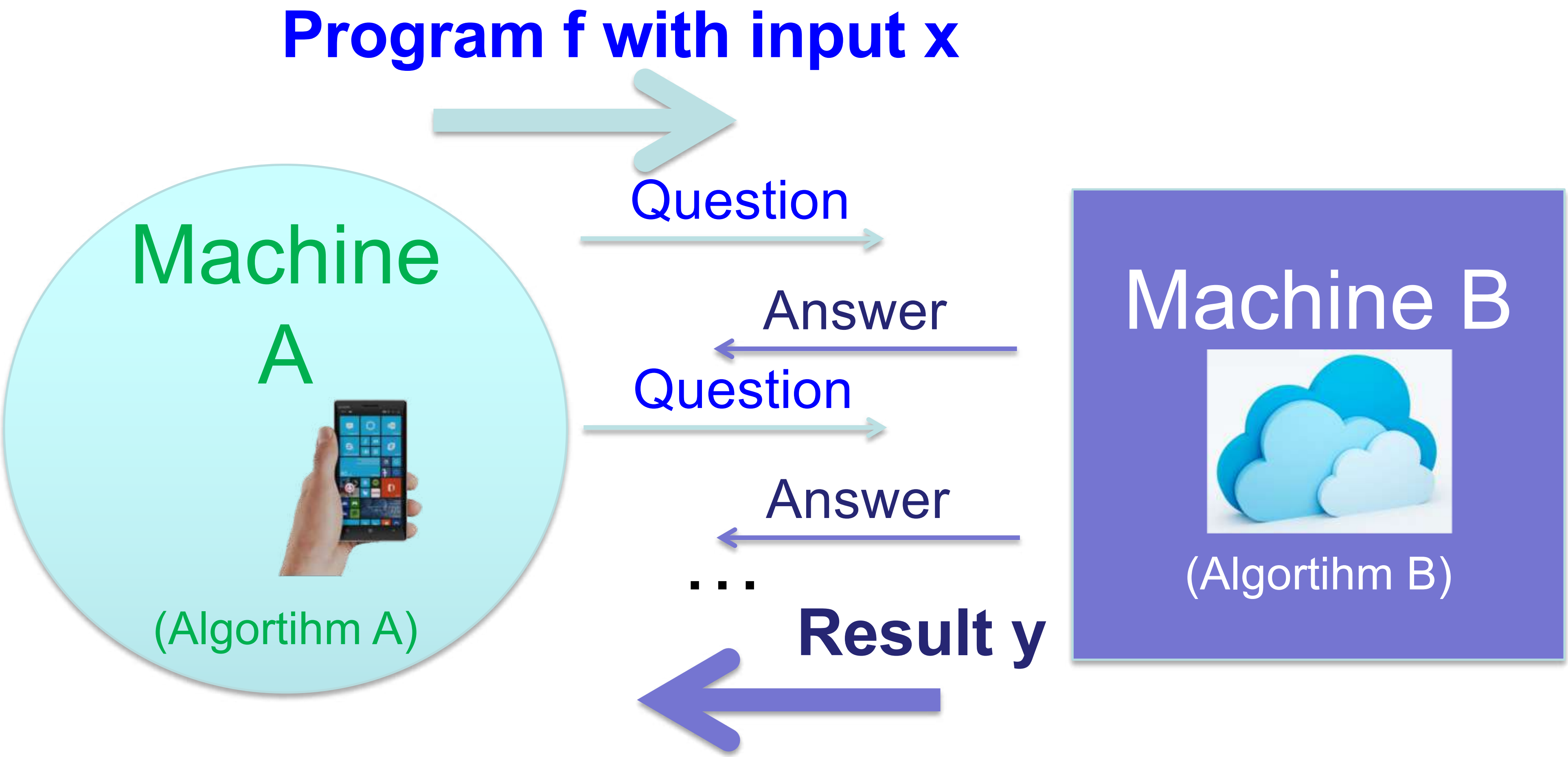
Jean-Louis Roch
Grenoble INP-Ensimag, Grenoble-Alpes University, France

1. Computation with encrypted data : FHE
2. Interactive verification of results
3. Zero-knowledge proofs
   - Interactive zero-knowledge protocols
   - exercise
4. Secure multiparty Computations

Distributed, heterogeneous

Various computing abilities and levels of trust

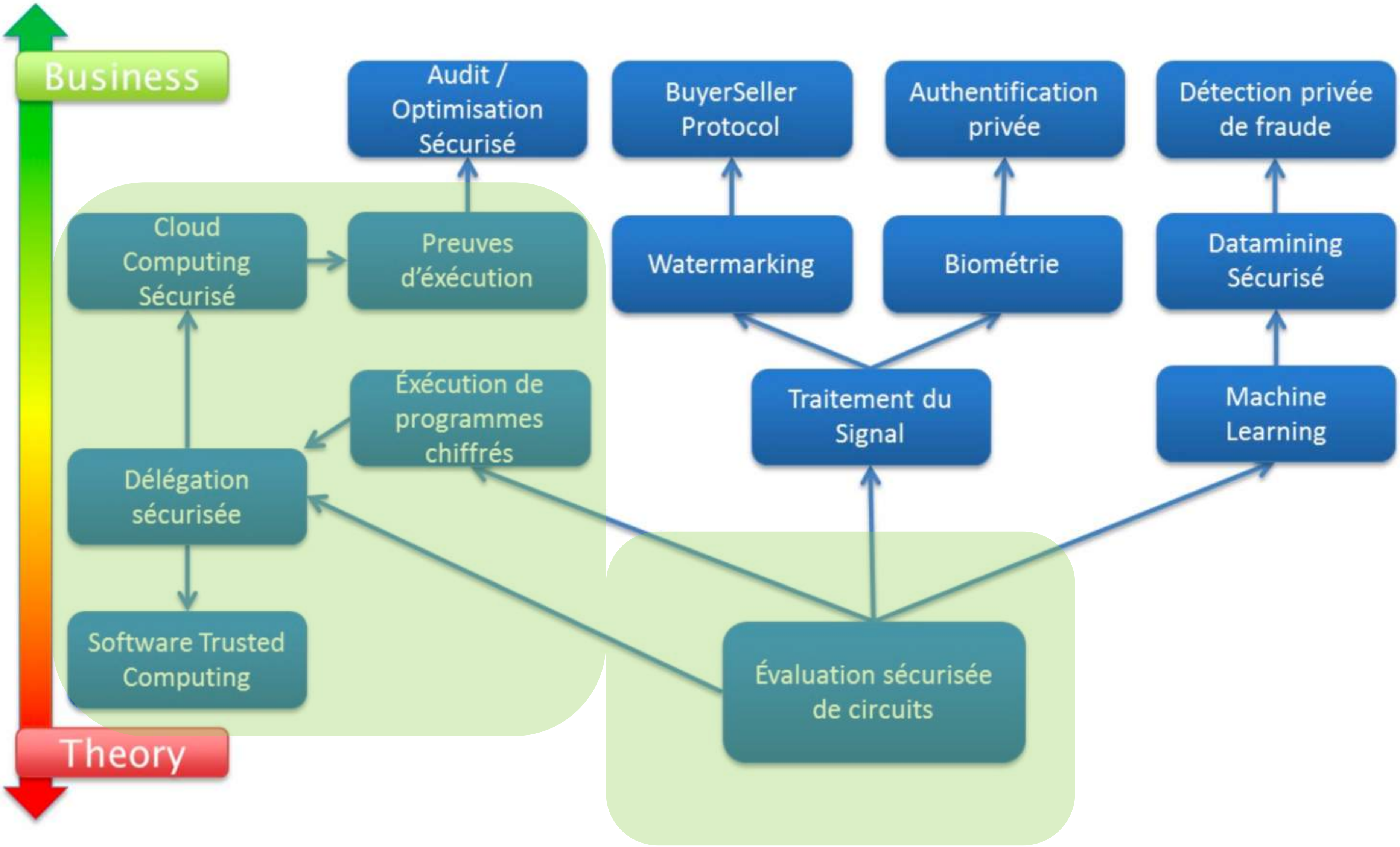
Outsourcing protocols and security



Trust in the result ?

=> *protocols for trustfully delegation of computations*

Positioning in current trends & market



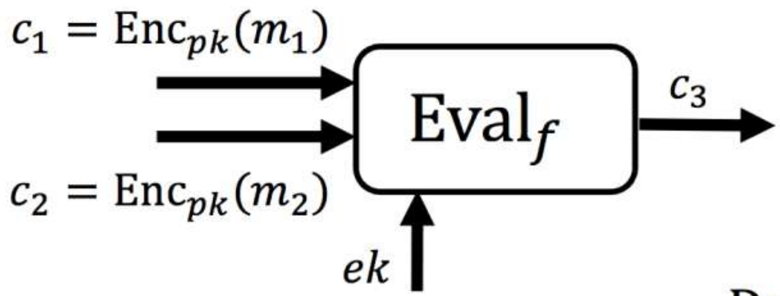
Outsourcing computations and security

Jean-Louis Roch
Grenoble INP-Ensimag, Grenoble-Alpes University, France

1. **Computation with encrypted data : FHE**
2. Interactive verification of results
3. Zero-knowledge proofs
   – Interactive zero-knowledge protocols
   – exercise
4. Secure multiparty computations

Computations with encrypted data

- Outsourcing computation with secret input
  - Computation is performed on encrypted data
  - Based on asymnetric encryption (eg thanks to fully **homomorphic encryption**) [Gentry 2009]



$$\mathrm{Dec}_{sk}\left(\mathrm{Eval}_f(ek, c_1, c_2)\right) = f(m_1, m_2)$$

Homomorphic multiplication

- **Remind El Gamal** (in cyclic group G with g a generator):
  - Bob has private key b and public key $B=g^b$
  - Alice: $c=E(m) = (c_1=g^r\ ,\ \ c_2=m.\ B^r)$ Bob= $D(c)=c_1^{-b}\ .\ c_2 = m$

- **El Gamal enables homomorphic multiplication:**
  - $C=E(M) = (g^r\ ,\ \ M.\ B^r\ )$
  - $C'=E(M')= (g^{r'}\ ,\ \ M'.B^{r'}\ )$
  - Multiplication of ciphertxt in G matches multiplication of plaintext *(e.g.integers plaintext)*
    $C.C' = (g^r.\ g^{r'},\ M.\ B^r.\ M'.\ B^{r'}\ ) = (g^{r+r'},\ M.M'.\ B^{r+r'})$
    $= E(\ M.M'\ )$ so Bob outsources M.M' without revealing M ad M'
  - Enables anyone to compute as many multiplications of ciphertexts as desired

  - **Question 1**: Does it work with RSA ?

Homomorphic encryption: El Gamal e-vote

- **Remind El Gamal** (in cyclic group G with g a generator):
  – Bob has private key b and public key $B=g^b$
  – Alice: $c=E(m) = (c_1=g^r\ ,\ c_2=m.\ B^r)$ Bob= $D(c)=c_1^{-b}\ .\ c_2 = m$

- **El Gamal enables homomorphic addition** *(note Alice encrypts g^M instead of M)*
  – $C=E(g^M) = (g^r\ ,\ g^M\ .\ B^r\ )$ (encode g^M instead of M)
  – $C'=E(g^{M'})= (g^{r'}\ ,\ g^{M'}\ .\ B^{r'}\ )$
  – Multiplication of ciphertxt in G matches addition of plaintext *(e.g.integers plaintext)*
  $C.C' = (g^r.\ g^{r'},\ g^M.\ B^r.\ g^{M'}.\ B^{r'}\ ) = (g^{r+r'},\ g^{M+M'}.\ B^{r+r'})$
  $= E(\ g^{M\ +M'}\ )$
  – Enables anyone to compute as many additions of ciphertexts as desired
  – **Question 1**: if M+M' small, how to decrypt M+M' from $g^{M\ +M'}$ without discrete log ?

- **Application: electronic vote by homomorhic addition of small integers**
  – Each voter (Alice) sends her encrypted vote v (0 or 1) to the voting machine (Bob) :
    - $C(0)=(\ g^r\ ,\ B^r)$ $C(1)=(g^{r'}\ ,\ g.B^{r'})$: each voter checks her encrypted vote is correctly stored
  – Each one can compute the encrypted score of the vote : $\Pi_{C\ voter}(C\ ) = (\ g^{\Sigma r}\ ,\ g^{\Sigma v}.B^{\Sigma r})$
  – The voting machine knows secret b : it computes $g^{\Sigma v}$ and publishes score Σv and Σr
    - **Question 2:** How the voting machine computes $\Sigma_{voters}v$ from $\Pi_{C\ voter}(C\ ) = (\ g^{\Sigma r}\ ,\ g^{\Sigma v}.B^{\Sigma r})$?
    - **Question 3** : How each voter verifies the result $\Sigma_{voters}v$?

Fully Homomorphic Encryption (FHE)

- Does there exist homomorphic boolean encryption ? => YES [Craig 2010]

- **Somewhat Fully Homomorphic Encryption** [Marten van Dijk, Craig Gentry, Shai Halevi, Vinod Vaikuntanathan]
  - Secret p : a large odd integer (eg thousands of digits)
  - For x in {0,1} : E(x) = pq +2.r + x

    With random q ~ million of digits and r ~ twenty digits (the ***noise***)
  - Knowing p: (E (x) mod p) mod 2 = (2.r+x) mod 2 = x
  - Without knowing p: E(x) give no information

- **Fully homomorphic with x and y booleans**:
  - E(x)+E(x')=p.(q+q') + 2(r+r ')+ x+x' => mod p mod 2 = x XOR x'
  - E(x).E(x')=p (pqq'+q(2r'+x')+q'(2r+x)) + 2(2rr'+rx'+r'x) + x.x'

    => mod p mod 2 = x AND x'
  - Beware : AND and XOR operations increase the noise
    - If noise r larger than p, decryption is impossible (eg if 2r = p.u+v then (E(x) mod p) mod 2 = (v+x) mod 2)
    - chooice of p and the q's large enough!

- Anyway with operations, the noise increases and may become larger than p
  Key Gentry's idea: *remote* ***bootstrapping***
  - Refresh the noise by outsourcing « mod p » on the cipher domain

    homomorphic computation with AND and XOR (so without revealing p, only its ciphering !)

- Many applications of FHE :
  - example: outsourcing AES encryption/decryption !

Outsourcing and privacy

- Homomorphic scheme enables to outsource encryption with secret key (or signature)

- Homomorphic encryption enables publicly Verifiable computation [Fiore, Gennaro 2012, …]
  - Server computes on private data and produces a verifiable digest of the computation

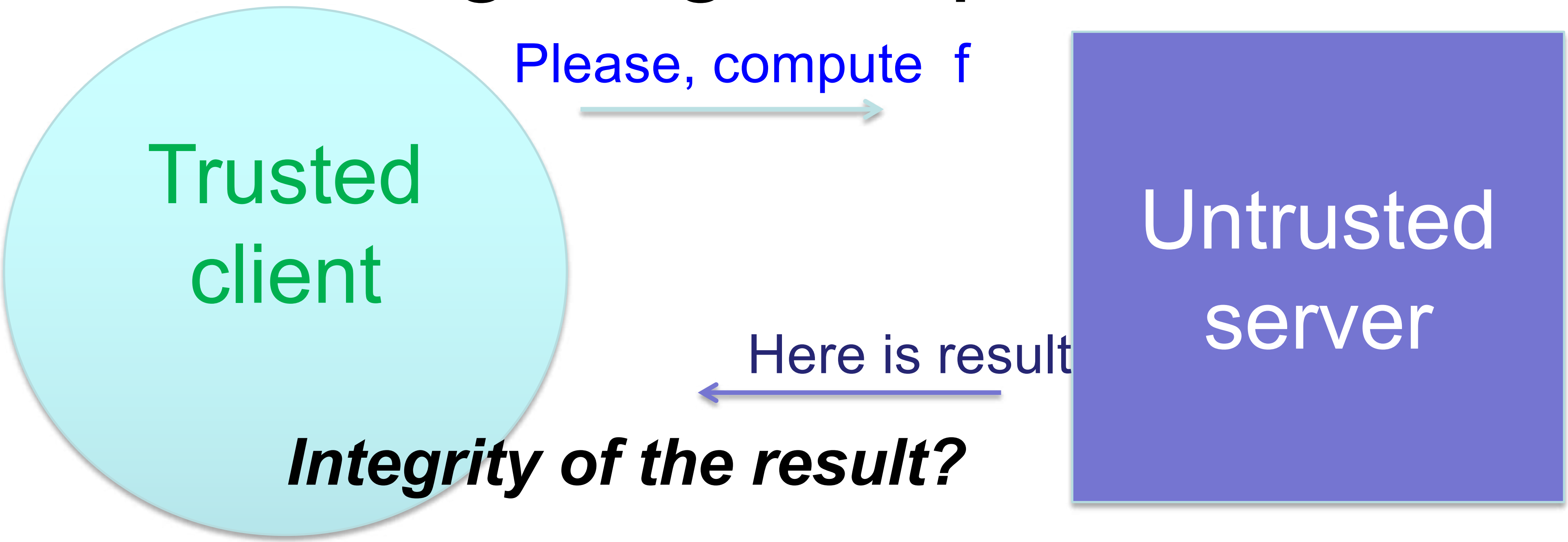
  - Enables some verification of the computation
    - Different from a direct result certification

Outsourcing computations and security

Jean-Louis Roch
Grenoble INP-Ensimag, Grenoble-Alpes University, France

1. Computation with encrypted data : FHE
2. **Interactive verification of results**
3. Zero-knowledge proofs
   - Interactive zero-knowledge protocols
   - exercise
4. Multiparty computations

Delegating computation

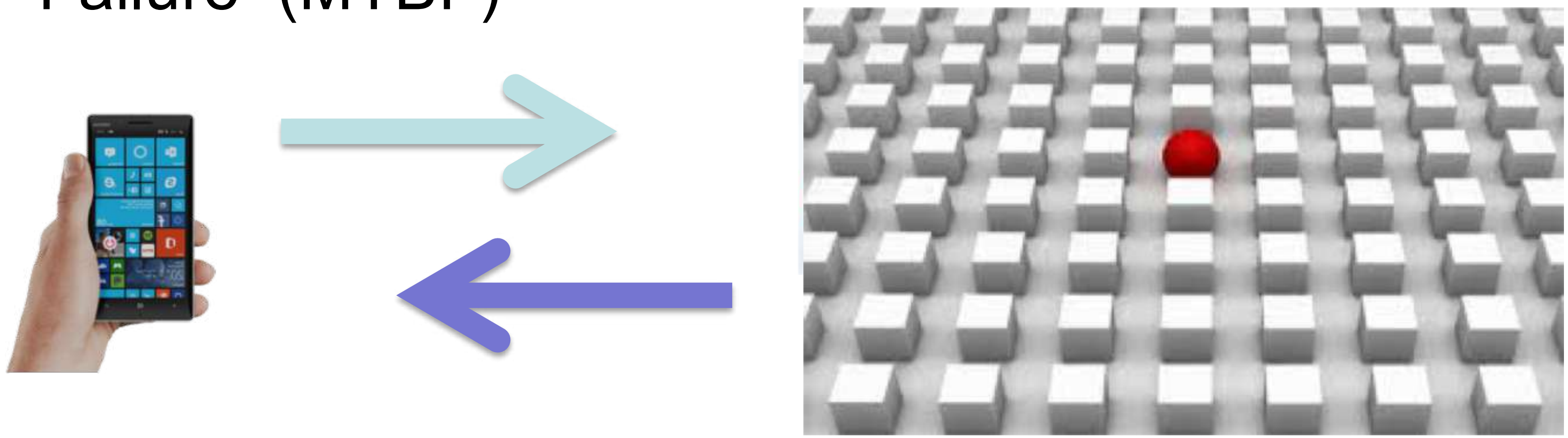


Integrity of the result?

- Contexts
  - Co-processor (overclocked…)
  - Supercomputer (soft errors)
  - Cloud computing
  - Volunteer computing

Attack models

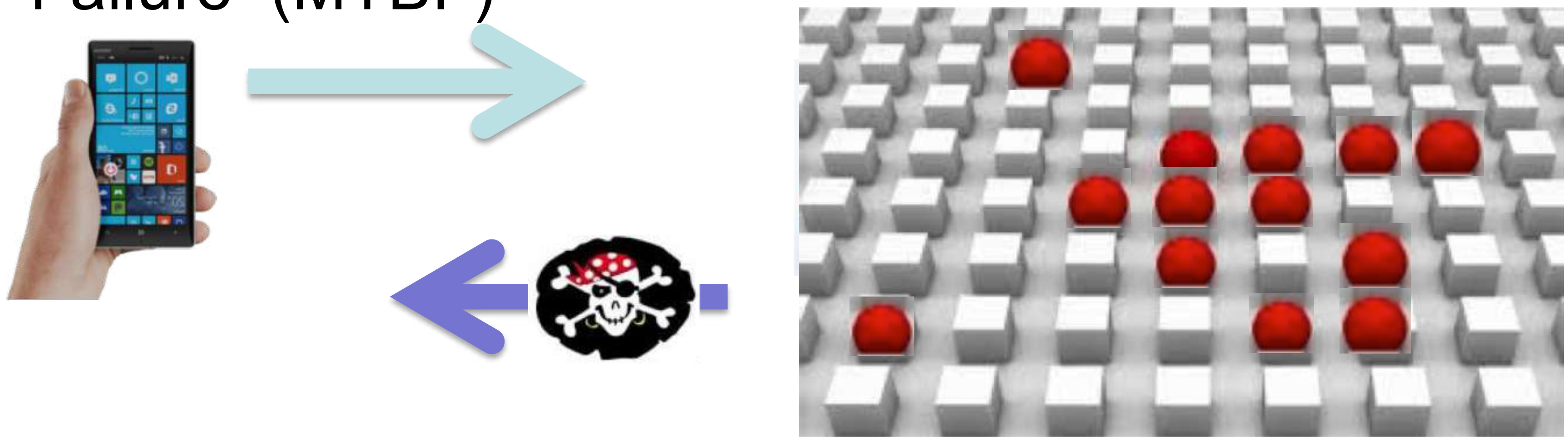
- No attack [current HPC and grid computing platform]
  - Failure (MTBF)



- **Attack on few isolated resources**
  - Soft errors - corruption of part of the computation

Attack models

- No attack [current HPC and grid computing platform]
  - Failure (MTBF)

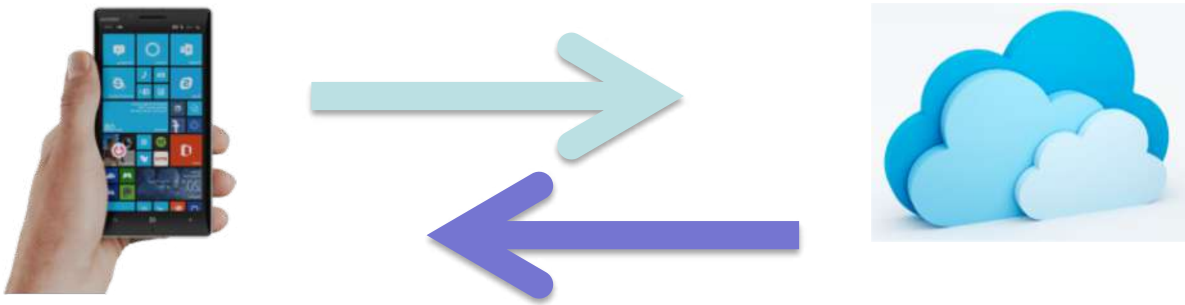


- **Attack on few isolated resources**
  - Soft errors - corruption of part of the computation
- **Massive attacks**

Countermeasures against such attacks (detect/correct)

Verifiable [outsourced] computation

- **Trusted but slow Client** (Verifier, *Victor*) sends a function F with input x to the server



- **Fast but untrusted Server** (Prover, *Peggy*) returns y = F(x) and a proof Π that y is correct.

Computing Π should take almost same time than F.

Verifying Π should take less time than computing F.

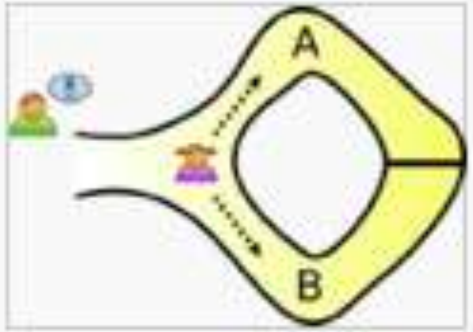
Motivating example

- Peggy has developed a nice application that efficiently solves Traveling Salesman Problem
- Victor sends Peggy the location (map) of his clients and pays her for the shortest Hamiltonian circuit
- Can Victor check he really gets the shortest?

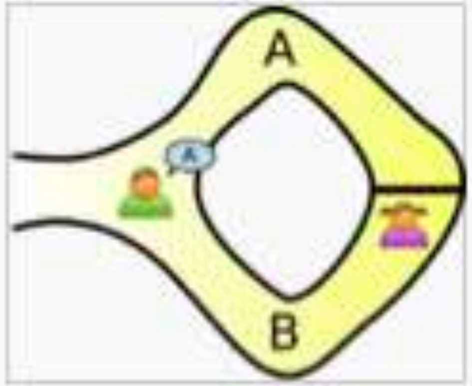
Example [wikipedia]

- A tunnel, closed by a trapdoor rock.
- Ali Baba knows the secret
  - « Iftah Ya Simsim » («Open Sesame»)
  - "Close, Simsim" («Close Sesame»).

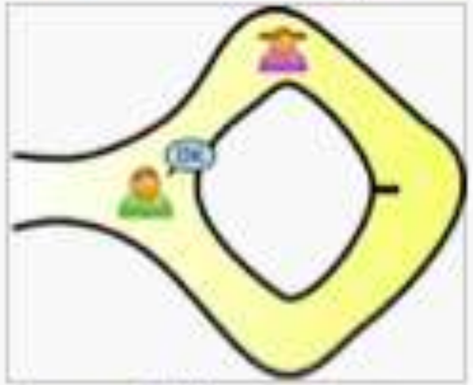
- Victor design a protocol that « proves » Ali Baba gets the secret without revealing it
  - Ali Baba (indeed Peggy) is *the Prover*
  - Victor is *the Verifier*
  - Peggy leaks no information (*0-knowledge*)



Peggy randomly takes either path A or B, while Victor waits outside



Victor chooses an exit path


Peggy reliably appears at the exit Victor names

Proof and Interactive proof

- Importance of « proof » in crypto: eg. identity proof=authentication
- Two parts in a proof:
  - Prover: knows the proof (-> the secret) [or is intended to know]
  - Verifier: verifies the proof is correct (-> authentication)

- Correctness of a proof system/verifier:
  - **Completeness**: ***every valid proof* is accepted** by the verifier
  - **Soundness**: <u>***every invalid proof***</u> **is rejected** by the verifier

- Interactive proof system
  - Protocol (questions/answers) between the verifier and the prover
  - Verifier: **probabilistic** algorithm, **polynomially bounded**
  - Soundness: every invalid proof is rejected with good probability ($\geq c > 1/2$)
  - Competeness: every valid proof is accepted with good probability ($\geq c' > 1/2$)

Decision problem
Does x belongs to L ?

- Verifier
  - An element x
  - Ask questions to prover to determine : « x ∈? L »
  - Gets anwer:
    - Completeness: Is convinced that x in L, if so
    - Soundess: reject « x in L » if not so

Fundamental theorem [Goldreich&al]

- Def: ***IP*** = set of decision problem that admits a randomized polynomial time verification algorithm
  i.e. both size of transcripts and nuber of operations performed by verifier are polynomial

- ***IP = PSPACE***
  - ***NP*** included in ***IP***.

- Any (PSPACE) computation admits a randomized determinstic polynomial verification algorithm.

Exercise: authentication based on quadratic residue (1/3)

- A **trusted third part T** provides a Blum integer n=p.q; n is public.

- **Alice (Prover) builds her secret and public keys:**
  - chooses a random s coprime to n
  - Compute $v:=(s^2)$ *mod* n. [NB v ranges over all square coprime to n]
    v = ***quadratic residue*** that admits s = ***modular square root***
  - Alice Secret key: s - Alice Public key: v and identity photo registered by T

- Naïve authentication protocol : **Bob (Verifier)** authenticates Alice as follows:
  1. Alice chooses a random r<n; she sends $y=r^2$ *mod* n and z=r.s *mod* n to Bob.
  2. Bob authenticates iff $z^2 = y.v \bmod n$.

- Question : does this protocol ensures :
  1. Completeness ? (ie anyone who knows the secret key s is accepted)
  2. Soundness ? (ie anyone who doesn't know the secret key s is rejected)

Exercise: authentication based on quadratic residue (2/3)

- A **trusted third part T** provides a Blum integer n=p.q; n is public.

- **Alice (Prover) builds her secret and public keys:**
  - chooses a random s coprime to n
  - Compute v:=(s^2) *mod* n. [NB v ranges over all square coprime to n]
    v = ***quadratic residue*** that admits s = ***modular square root***
  - Alice Secret key: s - Alice Public key: v and identity photo registered by P

- Authentication protocol : **Bob (Verifier)** authenticates Alice as follows:
  1. Alice chooses a random r<n; she sends y=r^2 *mod* n to Bob.
  2. Bob sends a uniformly random bit : b (ie b is either 0 or 1 with probability 1/2)
  3. Alice computes z := rs^b mod *n* and sends *z* to Bob.
     Bob authenticates iff z^2 = y.v^b mod n.

- Question : does this protocol ensures :
  1. Completeness ? (ie anyone who knows the secret key s is accepted)
  2. Soundness ? (ie anyone who doesn’t know the secret key s is rejected)

Exercise: authentication based on quadratic residue (3/3)

- A **trusted third part T** provides a Blum integer n=p.q; n is public.

- **Alice (Prover) builds her secret and public keys:**
  - For i=1, …, k: chooses at random s_i coprime to n
  - Compute $v_i := (s_i^2)$ *mod* n. [NB v_i ranges over all square coprime to n]
    v_i = ***quadratic residue*** that admits s_i = ***modular square root***
  - Secret key: $s_1, \ldots, s_k$
  - Public key: $v_1, \ldots, v_k$ and identity photo, … registered by T

- Authentication protocol : **Bob (Verifier)** authenticates Alice as follows:
  1. Alice chooses a random r<n; she sends $y=r^2$ *mod* n to Bob.
  2. Bob sends a random bits: $b_1, \ldots, b_k$
  3. Alice computes $z := r s_1^{b_1} . \ldots . s_k^{b_k} \bmod n$ and sends *z* to Bob.
     Bob authenticates iff $z^2 = y.v_1^{b_1}. \ldots .v_k^{b_k} \bmod n$.

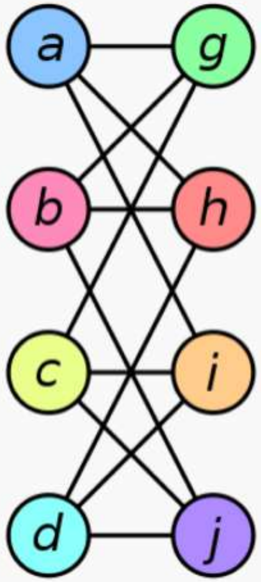
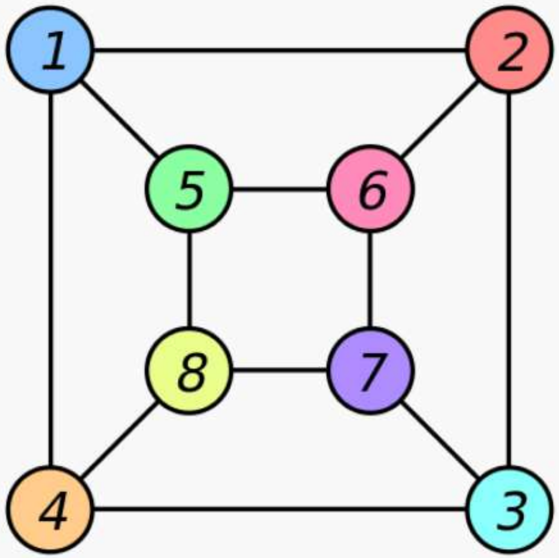
- Question : prove that this protocol provides:
  1. Completeness (ie anyone who knows the secret key s is accepted)
  2. Soundness with error probability $\leq 2^{-k}$
     (ie anyone who doesn't know the secret key s is rejected with proba $\geq 1- 2^{-k}$)

The power of interaction

Verifier
(Victor)

Prover
(Peggy)

https://en.wikipedia.org/wiki/Graph_isomorphism

Graph G	Graph H	An isomorphism between G and H
		$f(a) = 1$ $f(b) = 6$ $f(c) = 8$ $f(d) = 3$ $f(g) = 5$ $f(h) = 2$ $f(i) = 4$ $f(j) = 7$

On 2010/10/24, 8 am

- ∈ NP, but not known to be in NP or in NP-complete or in NP-intermediate
- Does it belongs to co-NP or not ? (Open question)
- but Subgraph isomorphism problem is NP-complete

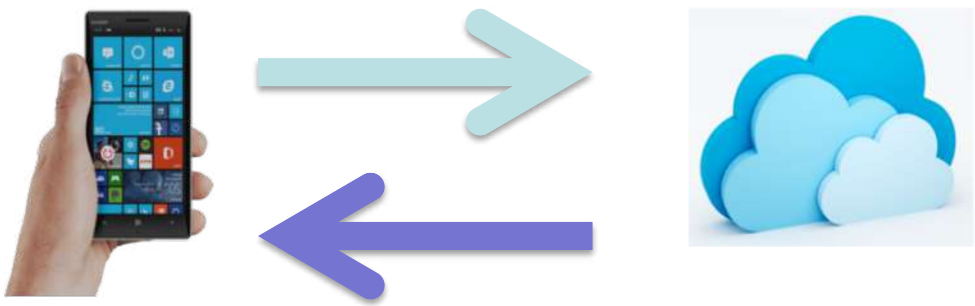
Example of interactive computation

- Graph isomorphism:
  - Input: G=(V,E) and G'=(V',E')
  - Output: YES iff G == G' (i.e. a permutation of V ->V' makes E=E')
- In NP but not known today to be NP-complete or in P
  - In 2015, Babai proposes a quasi-polynomial algorithm [2^O(log^k n)]
    (a bug was claimed on 2017/1/1 and fix on 2017/1/7)
- Not known to be in co-NP
- Assume an NP Oracle for Graph isomorphism =>
  then a probabilistic verifier can verifies that two graphs are not isomorphic in polynomial time.
  - Protocol and error probability analysis.

Interactive graph [non]-isomorphism

- Victor
  - Toss b := rand{1,2}

  - H := random_pemutation (E_b)
  - Asks Peggy: to which H is isomorphic to : E_1 or E_2 ?



Peggy returns y and Π

  - Victor checks Π and if OK
    - If y $\neq$b : Victor has a proof that E_1 isomorphic to E_2
    - Else y = b : Victor stated that E_1 is not isomorphic to E_2 with error probability ½

Interactive Algorithm Graph Isomorhism

Verifier

AlgoGraphIso($G_1=(V_1,E_1)$, $G_2=(V_2,E_2)$) {

If ($\#V_1$!= $\#V_2$) or ($\#E_1$!= $\#E_2$)

return "NO : G_1 not isomorphic to G2";

n := $\#V_1$;

For (i=1 .. k) {

P := randompermutation([1, …, n]) ;

b := random({1,2}) ;

G' := $P(G_b)$;

(i, P_i) := Call **OracleWhichIsIso**(G_1, G_2, G') ;

If ($G_i \neq P_i$ (G')) FAILURE("Oracle is not reliable") ;

If (b ≠ i) **return "YES : G_1 is isomorphic to G_2"** ;

}

return "NO : G_1 not isomorphic to G_2";

}

Prover

OracleWhichIsIso(G_1, G_2, G') {

// precondition: G' is isomorphic to

// G_1 or G_2 or both.

// Output: i into {1,2} and a permutation

// P_i such that G_i = P(G')

… ;

Return (i, P_i) ;

}

Theorem: Assuming OracleWhichIsIso of polynomial time, AlgoGraphIso(G_1, G_2) proves in polynomial time $k.n^{O(1)}$ that :

- either G_1 is isomorphic to G_2 (no error)
- or G_1 is not isomorphic with error probability $\leq 2^{-k}$.

Thus, it is a MonteCarlo (randomized) algorithm for proving GRAPH ISOMORPHISM

Analysis of error probability

***Truth:* $G_1 = G_2$??** \ ***Prob(Output of** AlgoGraphIso(G_1, G_2))*	***"YES : G_1 is isomorphic to G_2"***	***"NO: G_1 not isomorphic to G_2"***
Case $G_1 = G_2$ (completeness)	Prob = $1 - 2^{-k}$	Prob = 2^{-k}
No: Case $G_1 \neq G_2$ (soundness)	Impossible (Prob = 0)	Always (Prob = 1)

-When the algorithm output YES : G_1 is isomorphic to G_2 then $G_1 = G_2$
=> no error on this output.

-When the algorithm output "NO: G_1 not isomorphic to G_2" then we may have an error (iff $G_1 = G_2$), but with a probability $\leq 2^{-k}$

One-sided error => Monte Carlo algorithm for Graph-Isomorphism

Efficient verifiable computing by spot checking

- Check polynomial equality by random evaluation [Schwartz-Zippel]
  - Choose $r_1, \dots, r_n$ at random in a subset S of a field
  - If $Q(r_1, \dots, r_n)=0$ then $Q == 0$ with error probability $\leq \deg(Q) / \#S$

- Example: *Verifying matrix multiplication* (**Friedval's algorithm**)
  - To check C = A.B, choose a random vector r
    and verify C . r = A . (B . r)

    Cost : linear in size(A) + size(B) + size(C)

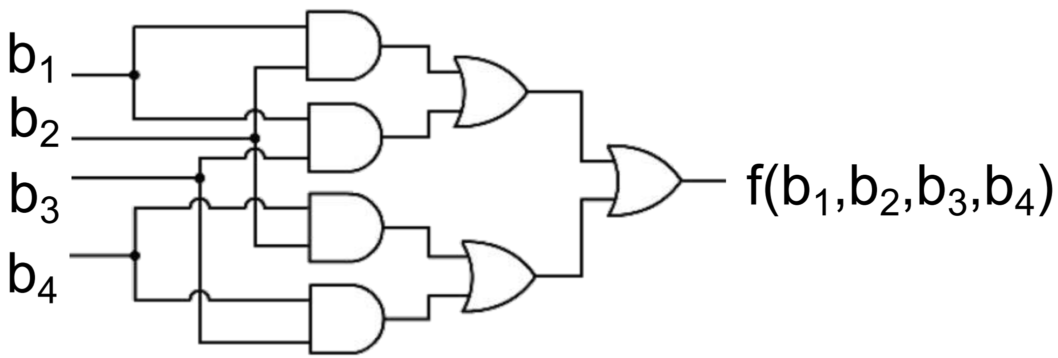
Interactive linear algebra

- Most dense linear algebra reduces to Matrix multiplication
  - Locally compute the (recursive) scheme in $O(n^2)$ while outsourcing all Matrix Multiplications
  - [Algorithm-Based Secure and Fault Tolerant Outsourcing of Matrix Computations, A Kumar, JL Roch, HAL 2013]

- Alternatively provide efficient certificates for sparse linear algebra
  - [Interactive certificates for linear algebra, JD Dumas, E Kaltofen, ISSAC 2014]

Verifying general circuits

- Inputs : $b_1 \dots b_n$ Outputs : $y_1 \dots y_m$
- How to verify $y_1 \dots y_m = f(b_1 \dots b_n)$



The power of interaction

- Theorem : IP = PSPACE
- Any problem in PSPACE has a polynomial verifier
  - TQBF (quantified Boolean formula problem)
- A polynomial interactive scheme for #SAT

P, NP, …. IP = PSACE

Complexity classes

Decision problems (1 output bit: YES/ NO)

Deterministic polynomial time:

- P : both Yes/No sides
- NP : certification for the Yes side
- co-NP: certification for the No side

Randomized polynomial time:

- BPP: Atlantic City: prob(error) < 1/2
- RPP: Monte Carlo: prob(error YES side)=0 ; prob(error NO side)< 1/2
- ZPP: Las Vegas: prob(failure)<1/2 but prob(error)=0

IP Interactive proof

- Verifier: randomized polynomial time
- Prover: interactive (dynamic), unbound power
  - F(x) = YES => it exists a correct prover Π such that Prob[Verifier (Π, x) accepts] = 1;
  - F(x) = NO => for all prover Π: Prob[Verifier (Π, x) accepts] < 1/2.
- Theorem: IP = PSPACE (interaction with randomized algorithms helps!)

PCP: Probabilistiic Checkable Proofs (static proof)

- PCP(r, q) : the verifier uses random bits and reads q bits of the proof only.
- Theorem: NP=PCP(log n, O(1))

#3-SAT in IP

- Arithmetization in F_2: each clause c has a poly. Q(c)
  - Q(not(x)) = 1-x Q(x and y) = x.y
  - Q(x or not(y) or z)=Q(not(not(x) and y and not(z))= 1–((1-x).y.(1-z))
- Let F = c_1 and … and c_m a 3-SAT CNF formula, and $g(X_1, \ldots, X_n) = Q(c_1).Q(C_2). \ldots .Q(c_m)$: deg(g) ≤ 3m
  Then $\#F = \Sigma_{b_1=0,1} \ldots \Sigma_{b_n=0,1}\, g(b_1, \ldots, b_n)$
- Since $\#F \leq 2^n$, for $p>2^n$, (#F=K) is equivalent to (#F=K mod p)
  - To limit to a polynomial number of operations, computation is performed mod a prime p in $2^n .. 2^{n+1}$ (provided by prover and checked by verifier)
- Let $h_n(X_n) = \Sigma_{b_1=0,1} \ldots \Sigma_{b_{n-1}=0,1} g(b_1, b_2, \ldots, b_{n-1}, X_n)$:
  h_n is an univariate polynomial (in X_n) of degree ≤ m

#3-SAT: interactive polynomial proof

Verifier

input: $F(X_1, \ldots, X_n) = (c_1 \text{ and } \ldots \text{ and } c_m)$
K an integer; let $g(x) = \Pi_{i=1,n} Pol(c_i)$

Accepts iff convinced that #F = K.

Preliminar receive p, check p is prime in $\{2^n, 2^{2n}\}$

Compute $g(X_1, \ldots, X_n) = \Pi_{i=1,n} Pol(c_i)$ $\deg(g) \leq 3m$

Check $K = \Sigma_{X1=0,1} \ldots \Sigma_{Xn=0,1}\, g(X_1, \ldots, X_n)$ [p] :

1. If n=1, if (g(0)+g(1) = K) accept ; else reject.
   If n≥2, ask $h_n(X)$ to P.

3. Receive s(X) of degree ≤m.
   Compute v=s(0)+s(1); if (v ≠ K) reject.
   Else choose r=random(0, … p-1); let $K_n = s(r)$
   and use the same protocol to check
   $K_n = \Sigma_{X1=0,1} \ldots \Sigma_{Xn-1=0,1}\, g(X_1, \ldots, X_{n-1}, r)$ [p]

Prover

Preliminar: sends p prime in $\{2^n, 2^{2n}\}$

2. Send s(X) ; [note that if P is not cheating, $s(X) = h_n(X)$]

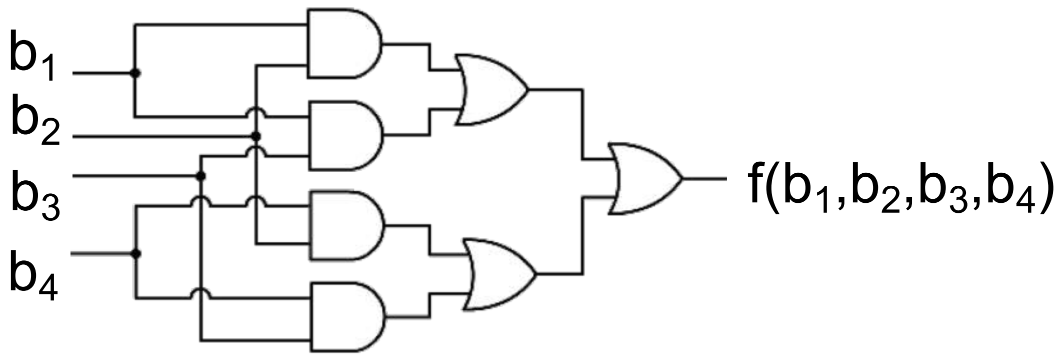
Theorem: This is a sound and complete, polynomial time randomized interactive proof of #3-SAT.

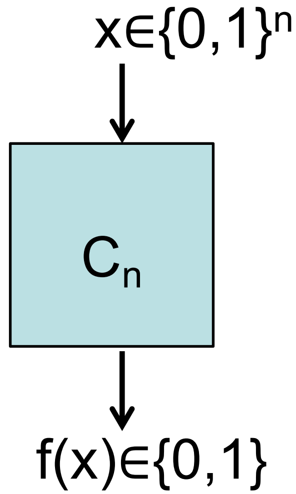
Moreover, prob(V rejects | K ≠ #F) ≥ (1-m/p)^n ,
also prob(error) ≤ 1-(1-m/p)^n ≤ $mn2^{-n}$.

[Lund, Fortnow, Karloff, Nisan 1992]

A key tool: the **sum-check** protocol

- **Input** : a (boolean) circuit C_n of depth δ that implements a function f with n bits in input :





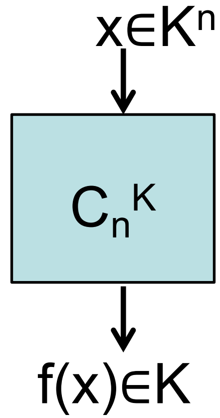
- **Output** : $S_n = \Sigma_{b_1=0,1} \ldots \Sigma_{b_n=0,1} f(b_1, \ldots, b_n)$

- Let $d=2^{\delta}$: #usefull gates $\leq$ d.
  Theorem: The verifier *interactively* computes S_n in polynomial time $(n+d)^{O(1)}$. (if δ=O(log n), polynomial in n)

- Application: number of elements that verify a predicate (#SAT)

Key 1: Arithmetization

- Transform the boolean circuit C_n in an arithmetic circuit C_n^2 in any field K (eg mod *p*) :
    - x *and* y = $x \cdot_K y$ *not*(x) = 1 – x
    - x *or* y = not (not(x) *and* not(y)) = $1 -_K (1 -_K x) \cdot_K (1 -_K y)$
- Transform the circuit C_n^2 in a circuit C_n^K with input in a (large) field K.
    - Gates are + and x in K
    - When inputs are 0 or 1, the output is the same than C_n



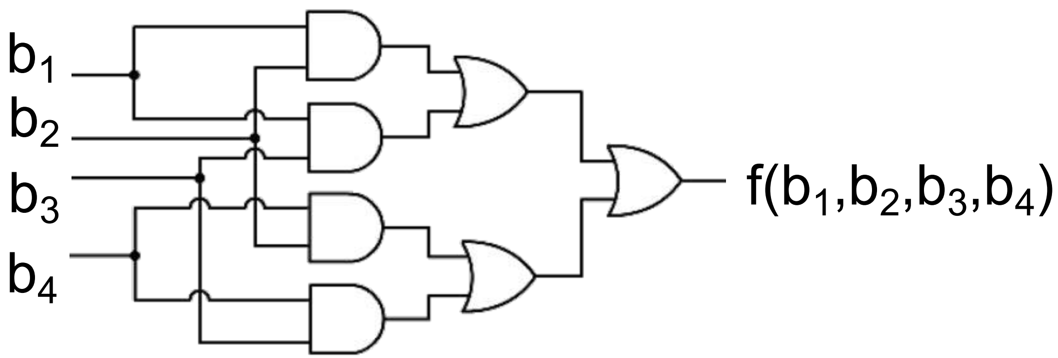
- Now, the circuit can be seen as a polynomial in n variables (the input) with degree d
    - For m=log #K, the circuit can be evaluated in time $(nm)^{O(1)}$, polynomial for any [random] input in K^n.
- **Key 2: induction on the number of sum**
    - Each sub-sum is verified with Schwartz-Zippel

Interactive verification of #3-SAT

- Let: Φ = (c_1 and … and c_m) be a 3-SAT CNF formula
- Arithmetization of Φ gives $g(X_1, \ldots, X_n) = Q(c_1).Q(c_2)\ldots Q(c_m)$
- $\mathrm{Deg}(g) \leq 3m$ (small)
  Polynomial-size circuit to evaluate g at any $(b_1, \ldots, b_n)$
- To prove #SAT(Φ)=K reduces to a sequence of sum-check
  $$\Sigma_{b_1=0,1} \ldots \Sigma_{b_n=0,1}\, g(b_1, \ldots, b_n)$$
  – computation in F_p with p prime $> 2^n$

Verifying general circuits

- Inputs : $b_1 \ldots b_n$ Outputs : $y_1 \ldots y_m$
- How to verify $y_1 \ldots y_m = f(b_1 \ldots b_n)$



[Goldwasser, Kalai, Rothblum 2008][Thaler Crypto 2015]

Outsourcing general circuits

- Circuits C with n inputs and outputs,
  - Work W, depth D
  - Each level is of degree 1 (multilinear extension)
- Computation is valid iff all levels are corrects
  - Verified by a sum-check at each level

- Cost = $(N + D) \log^{O(1)} (N + W)$

- Optimization when the computation resumes to a reduction of independent parallel computations

Illustration on Matrix Multiplication

- Let A and B matrices (n,n) in K with $m = \log_2 n$
- A is a (boolean) function $\{0,1\}^m x \{0,1\}^m x \to K$:

  $$A(i_1, \ldots i_m, j_1, \ldots, j_m) = A(i,j)$$

- Let g_A be the polynomial multilinear extension of A
- The g_C verifies

  $$g_C(i_1, \ldots i_m, j_1, \ldots, j_m) = \sum_{k=0..n} g_A(i_1, \ldots i_m, k_1, \ldots, k_m).g_B(k_1, \ldots k_m, j_1, \ldots, j_m)$$

- With the sum-check protocol, this sum of *n* elements is verified in O(log *n*)
- Generalizes to parallel compuations with logarithmic depth (NC1)

Practical efficiency ?

- Further improvements [Thaler]
  - Sum of products only
  - Same circuit for any coefficient

Problem Size	Naïve MatMult Time	Additional P time	V Time	Rounds	Protocol Comm
1024 x 1024	2.17 s	0.03 s	0.67 s	11	264 bytes
2048 x 2048	18.23 s	0.13 s	2.89 s	12	288 bytes

  - Yet far from Fiedvald's verification

What have we learned ?

- Interactive proof : generalization of a mathematical proof in which a prover interacts with a polynomial-time probabilistic verifier:
  - Completeness and soundness
- Input: x,   proof of property L(x)
  Correct proof: x is accepted iff L(x) is true.
  - Completeness : any x: L(x)=true is accepted (with prob≥2/3).
  - Soundess : any y: L(y)=false is rejected (with prob≥2/3).

- Powerful interactive proof w.r.t. « static » proof
  - IP = PSACE

Conclusion on outsourcing

- Verifying delegated computation
  - Interaction between models provides power
  - Enables the provable use of untrusted platforms
    - Overclocked processors, algorithms with faults, quantum computing, …
  - Fully Homomorphic Encryption (powerful but yet expensive)
  - Current research to improve FHE efficiency

- On going research - Applications
  - Cloud computing. (web services)
  - Outsourced fault-tolerant computation
  - Secure remote storage (privacy)
  - Secure control-command for critical infratscture (SCADA)
  - A promising market (eg digital doctor)

https://www.youtube.com/watch?v=1MCa4d00OLQ

Outsourcing computations and security

Jean-Louis Roch
Grenoble INP-Ensimag, Grenoble-Alpes University, France

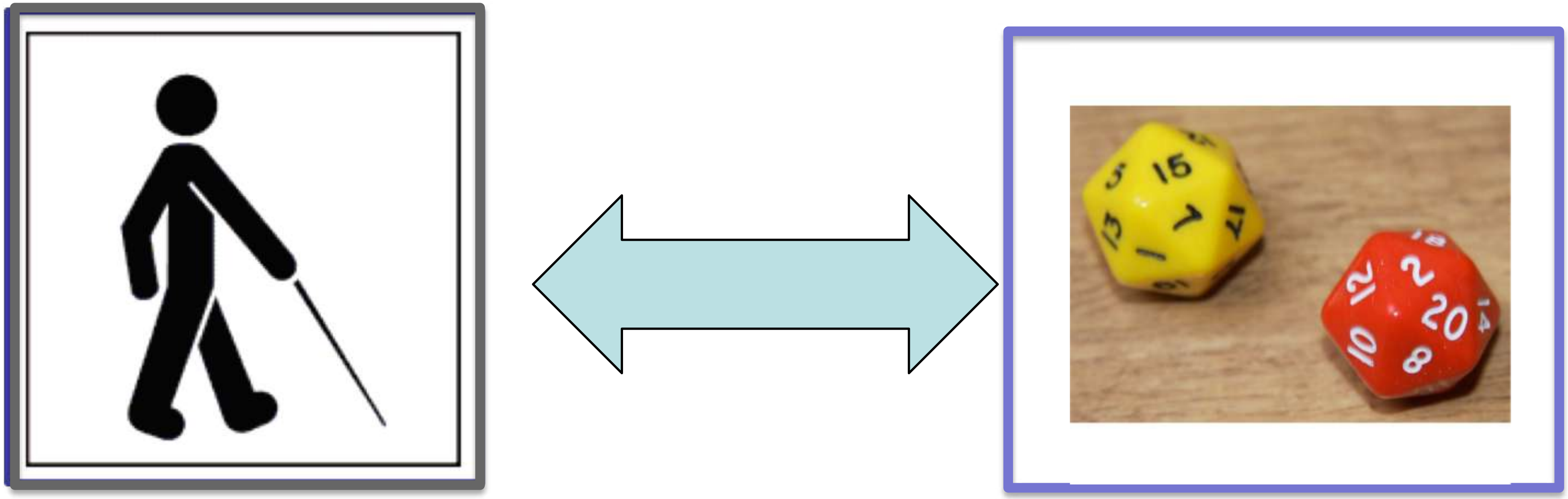
1. Computation with encrypted data : FHE
2. Interactive verification of results
3. **Zero-knowledge proofs**
   – Interactive zero-knowledge protocols
   – exercise
4. Secure multiparty computations

Interactive proof and zero knowledge protocols

- Zero-knowledge: definition
- Probabilistic complexity classes and Interactive proofs
  - Graph isomorphism and PCP
- Some zero knowledge protocols:
  - Feige-Fiat-Shamir authentication protocol
  - Extension to signature
  - Guillou-Quisquater authentication and signature

- Computational Complexity: A Modern Approach. Sanjeev Arora and Boaz Barak http://www.cs.princeton.edu/theory/complexity/
- Handbook of Applied Cryptography [Menzenes, van Oorschot, Vanstone]
- Applied Cryptography [Schneier]
- Contemporary cryptography [Opplinger]

The power of interaction



Verifier
(Victor)

Prover
(Peggy)

Zero knowledge

- How to state that the prover leaks *no information ?*

all interactive informations provided by the prover (ie the trasncripts) could have been produced offline by the verifier himself alone!

=> by stating the verifier can produce the transcript of the protocol in (expected) polynomial time alone, with no help of the prover !

- **Def:** a sound and correct interactive protocol is **zero-knowledge** if there exists a *non-interactive randomized polynomial time* algorithm (named « **simulator** ») which, for any input x accepted by the verifier (using interaction with the prover) can produce transcripts indistinguishable from those resulting from interaction with the real prover.

- **Consequence:** releases no information to an observer.

Graph [non]-isomorphism and zero knowledge

- In a zero-knowledge protocol, the verifier learns that G_1 is isomorphic to G_2 but nothing else.

Previous protocol (slide 24 or next) **not known to be zero-knowledge:**

correct transcript $X=(G', i, P')$ with $G'=P_{rand}(G_{rand})$ and $G_i= P'(G')$

- If $G_1 \neq G_2$: (we have b=i) => Entropy(transcript X) = 1 + log n!
  Simulation: $(P'^{-1}(G_i), i=rand(1,2), P'=RandPerm) ==_{distribution} X$
  => No information revealed !

- If G_1 is isomorphic to G_2 : Prover sends the permutation P_i such that $G_1= P_i(G_2)$: then i is independent form G'
  Entropy(transcript X) = 2 + log n!
  so the verifier learns 1 additional bit to only a random bit and a random permutation

Non-known zero knowledge Interactive Algorithm Graph Isomorhism

Verifier

AlgoGraphIso($G_1=(V_1,E_1)$, $G_2=(V_2,E_2)$) {

If ($\#V_1$!= $\#V_2$) or ($\#E_1$!= $\#E_2$)

return "NO : G_1 not isomorphic to G2";

n := $\#V_1$;

For (i=1 .. k) {

P := randompermutation([1, …, n]) ;

b := random({1,2}) ;

G' := P(G_b) ;

(i, P_i) := Call **OracleWhichIsIso**(G_1, G_2, G') ;

If ($G_i \neq P_i$ (G')) FAILURE("Oracle is not reliable") ;

If (b ≠ i) **return "YES : G_1 is isomorphic to G_2"** ;

}

return "NO : G_1 not isomorphic to G_2";

}

Prover

OracleWhichIsIso(G_1, G_2, G') {

// precondition: G' is isomorphic to

// G_1 or G_2 or both.

// Output: i into {1,2} and a permutation

// P_i such that G_i = P(G')

… ;

Return (i, P_i) ;

}

Theorem: Assuming OracleWhichIsIso of polynomial time, AlgoGraphIso(G_1, G_2) proves in polynomial time $k.n^{O(1)}$ that :

- either G_1 is isomorphic to G_2 (no error)
- or G_1 is not isomorphic with error probability $\leq 2^{-k}$.

Thus, it is a MonteCarlo (randomized) algorithm for proving GRAPH ISOMORPHISM

A zero-knowledge interactive proof for Graph Isomorhism

Verifier

input: ($G_1=(V_1,E_1)$, $G_2=(V_2,E_2)$)

Accepts prover if convinced that G1 is isomorphic to G2

2. Receives H;

   Chooses b=random(1,2) and sends b to the prover

4. receives P'' and checks H = P''(G_b)

Proover

gets G_1, G_2

private secret perm. P_s: $G_2=P_s(G_1)$

1. Chooses a random perm. P' and sends to verifier H=P'(G_2)

3. Receives b;

   if b=1 sends P''=P'oP_s to the verifier

   else b=2: sends P''=P' to the verifier

Theorem: This is a zero-knowledge, sound and complete, polynomial time interactive proof that the two graphs G_1 and G_2 are isomorphic.

Zero-knowledge interactive proof for Graph Isomorhism

- Completeness
- Soundness
- Zero-knowledge
- Polynomial time

Zero-knowledge interactive proof for Graph Isomorhism

- Completeness
  - if $G_1=G_2$, verifier accepts with probability 1.
- Soundness
  - if $G_1 \neq G_2$, verifier rejects with probability ≥ ½
- Zero-knowledge
  - Simulation algorithm:
    1. Choose first b=rand(1,2) and π random permutation (like P');
    2. Compute $H = \pi(G_b)$;
    3. Output transcript [H, b, π] ;
  - The transcript [H, b, π] is distributed uniformly, exactly as the transcript [H, b, P'] in the interactive protocol.
- Polynomial time

Another simulation algorithm
(following the prover's protocol but cheating)

Simulator:

Do {

1. b' = random(1,2) and π=random(permutation)
2. Compute $H=\pi(G_{b'})$ *// prover would send H to verifier*
3. b = random(1,2) ; *// prover would receive b from verifier*

} while (b ≠ b') ; *// cheat to find a valid transcript in polytime*

Output transcript [H, b, π]

- Polynomial time:
  – Expectation time = $\text{Time}_{\text{Loop\_body}} \cdot \sum_{k \geq 0} 2^{-k} \leq 2.\text{Time}_{\text{Loop\_body}}$

Exercise

- N is a public integer.
  Provide an interactive polynomial time protocol to prove a verifier that you know the factorization N=P.Q without revealing it.
  - Application:
    - a sensitive building, authorized people know 2 secret primes P and Q (and N=PQ)
    - The guard knows only N

Quadratic residue authentication: is this version **perfectly** zero-knowledge?

- A **trusted part T** provides a Blum integer n=p.q; n is public.

- **Alice (Prover) builds her secret and public keys:**
  – For i=1, …, k: chooses at random s_i coprime to n
  – Compute $v_i := (s_i^2)$ *mod* n. [NB v_i ranges over all square coprime to n]
  v_i = ***quadratic residue*** that admits s_i = ***modular square root***
  – Secret key: s_1 , …, s_k
  – Public key: v_1 , …, v_k and identity photo, … registered by T

- **Bob (Verifier)** authenticates Alice: **Zero-knowledge protocol in 3 messages** :
  1. Alice chooses a random r<n; she sends $y=r^2$ *mod* n to Bob.
  2. Bob sends k random bits: b_1 , …, b_k
  3. Alice computes $z := r s_1^{b_1} . \ldots . s_k^{b_k}$ mod *n* and sends *z* to Bob.
     Bob authenticates iff $z^2 = y . v_1^{b_1} . \ldots . v_k^{b_k}$ mod n.

- **Simulation algorithm** : *is the protocol perfectly zeo-knowledge?*
  1. Choose k random bits b_1 , …, b_k and a random z<n;
     compute $w = v_1^{b_1} . \ldots . v_k^{b_k}$ mod n and $y = z^2 . w^{-1}$ mod n ;
  2. Transcript is [y ; b_1 , …, b_k ; z]

Feige-Fiat-Shamir zero-knowledge authentication protocol

- A **trusted part T** computes a Blum integer n=p.q; n is public.

- **Alice (Prover) builds her secret and public keys:**
  - For i=1, …, k: chooses at random s_i coprime to n
  - Compute $v_i := (s_i^2)$ *mod* n. [NB v_i ranges over all square coprime to n]
    v_i = ***quadratic residue*** that admits s_i = ***modular square root***
  - Secret key: s_1 , …, s_k
  - Public key: v_1 , …, v_k and identity photo, … registered by T

- **Bob (Verifier)** authenticates Alice: **Zero-knowledge protocol in 3 messages** :
  1. Alice chooses a random r<n and a sign $u=\pm 1$; she sends $y=u.r^2$ *mod* n to Bob.
  2. Bob sends k random bits: b_1 , …, b_k
  3. Alice computes $z := r.\ s_1^{b_1}.\ \ldots\ .\ s_k^{b_k}$ mod *n* and sends *z* to Bob.
     Bob authenticates iff $z^2 = +/- \ y.v_1^{b_1}.\ \ldots\ .v_k^{b_k}$ mod n.

- Remark: possible variant: Alice chooses its own modulus n

Feige-Fiat-Shamir

Truth: **X=Alice or anyone else?** / *Prob(Output of authentication)*	***YES: "Authentication of Alice OK"***	***NO: "Authentication of Alice KO »***
Case X = Alice (completeness)	Always	Impossible
Case X ≠ Alice (soundness)	Prob = 2^{-k}	Prob = $1 - 2^{-k}$

- **Completeness**
  - Alice is allways authenticated (error prob=0)
- **Soundness**
  - Probability for Eve to impersonate Alice = 2^{-k}. If t rounds are performed: 2^{-kt}
- **Zero-knowledge**
  - A simulation algorithm exists that provides a transcript which is indistinguishable with the trace of interaction with correct prover.

From zero-knowledge authentication to zero knowledge signature

- Only one communication: the message+signature
  - The prover uses a CSPRNG (e.g. a secure hash function) to generate directly the random bits of the challenge
  - The bits are transmitted to the verifier, who verifies the signature.

- Example: Fiat-Shamir signature
  - Alice builds her secret key $(s_1, \ldots, s_k)$ and public key $(v_1, \ldots, v_k)$ as before.
  - Let M be a message Alice wants to sign.
  - Signature by Alice
    1. For i=1, …, t: chooses randomly r_i and computes w_i s.t. $w_i := r_i^2 \bmod n$.
    2. Computes $h = H(M \| w_1 \| \ldots \| w_t)$ this gives k.t bits b_{ik}, that appear as random (similarly to the ones generated by Bob in step 2 of Feige-Fiat-Shamir)
    3. Alice computes $z_i := r_i . s_1^{b_{i1}} . \ldots . s_k^{b_{ik}} \bmod n$ (for i = 1 .. t) ;

       She sends the message M and its signature: $\sigma = (z_1 \ldots z_t, b_{11} \ldots b_{tk})$ to Dan
  - Verification of signature σ by Dan:
    1. Dan computes $y_i := z_i^2 . (v_1^{b_{i1}} . \ldots . v_k^{b_{ik}})^{-1} \bmod n$ for i=1..t
       A correct signature gives $y_i = w_i$
    2. Computes $H(M, \| y_1 \| \ldots \| y_t)$ and
       he verifies that he obtains the bits b_{ik} in Alice's signature

Zero-knowledge vs other asymetric protocols

- No degradation with usage.
- No need of encryption algorithm.
- Efficiency: often higher communication/computation overheads in zero-knowledge protocols than public-key protocols.
- For both , provable security relies on conjectures (eg: intractability of quadratic residuosity)

Exercise

- Guillou-Quisquater zero-knowledge authentication and signature protocol.

Feige-Fiat-Shamir zero-knowledge authentication protocol

- A **trusted part T** (or Alice) computes a Blum integer n=p.q; n is public.
- **Alice (Prover) builds her secret and public keys:**
  - For i=1, …, k: chooses at random s_i coprime to n and n random bits d_i
  - Compute $v_i := (s_i^2)$ *mod* n. [NB v_i ranges over all square coprime to n]
    $(-1)^{d_i} v_i$ = ***quadratic residue*** that admits s_i = ***modular square root***
  - Secret key: $s_1, \ldots, s_k$. (Note that $v_i . s_i^2 = (-1)^{d_i}$ = 1 or -1 mod n)
  - Public key: $v_1, \ldots, v_k$ and identity photo, … registered by T

- **Bob (Verifier)** authenticates Alice: **Zero-knowledge protocol in 3 msgs** :
  1. Alice chooses a random value r < n. She sends $y := r^2$ *mod* n to Bob.
  2. Bob sends k random bits: $b_1, \ldots, b_k$
  3. Alice computes $z := r.\, s_1^{b_1} . \ldots . s_k^{b_k}$ mod n and sends z to Bob.
     Bob computes $w = z^2 . v_1^{b_1} . \ldots . v_k^{b_k}$ and authenticates iff y=w or y=-w mod n.

- Soundness and completeness, perfectly zero knowledge
  - Probability for Eve to impersonate Alice = 2^{-k}. If t rounds are performed: 2^{-kt}
  - Alice always authenticated (error prob=0)
  - Zero knowledge: transcript

Interactive zero knowledge protocol

What have we learned?

- Soundness + completeness
- Interactive proof (computers, profs) >> static proof (books)
- Zero-knowledge: simulation that provides a transcript indistinguishable from the correct interaction!
- Everywhere in crypto:
  - Authentication, signature, security proofs (IND-CCX)
- Perspective: outsourcing with verifiable trust

Outsourcing computations and security

Jean-Louis Roch
Grenoble INP-Ensimag, Grenoble-Alpes University, France

1. Computation with encrypted data : FHE
2. Interactive verification of results
3. Zero-knowledge proofs
   – Interactive zero-knowledge protocols
   – exercise
4. **Secure multiparty computations**

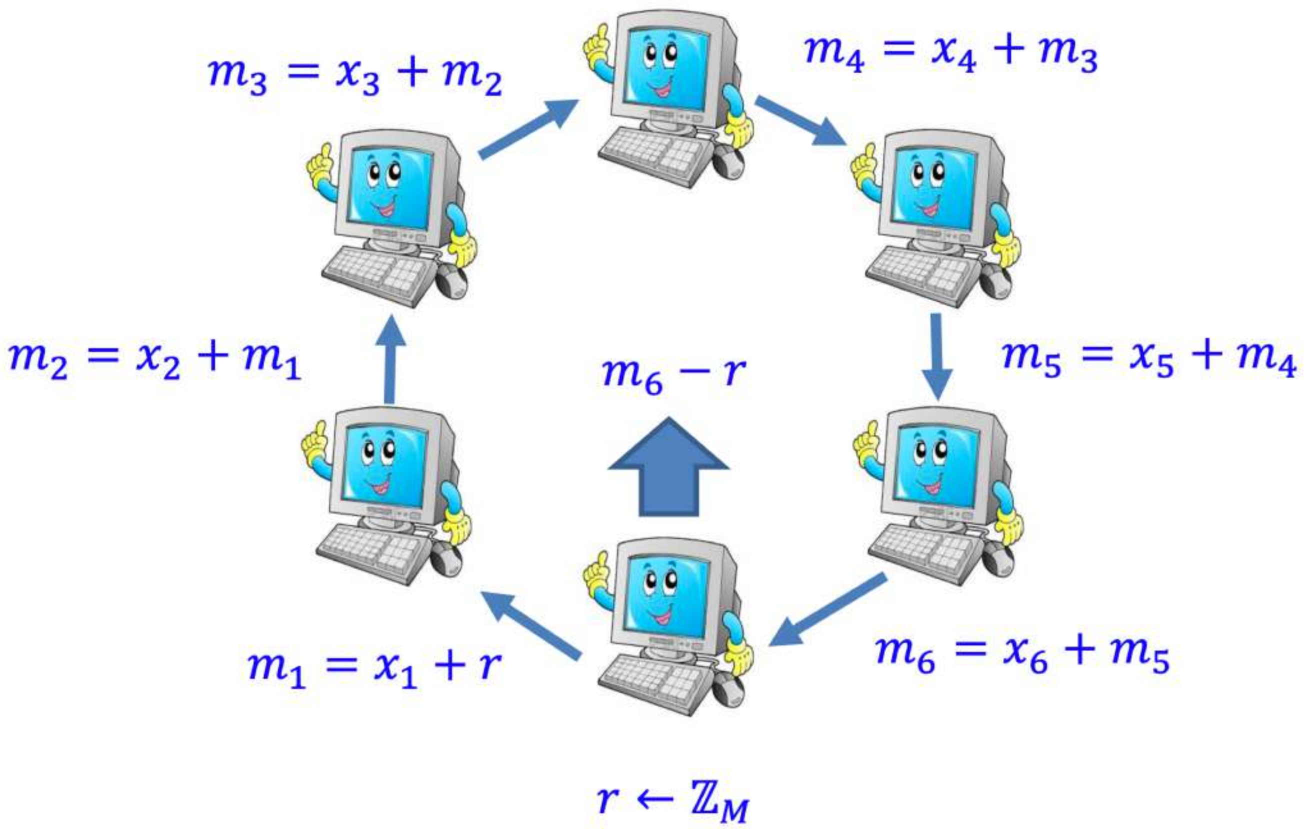
Secure multiparty computation

- Examples [Ran Cohen lecture : https://www.cs.tau.ac.il/~iftachh/Courses/Seminars/MPC/Intro.pdf]
- n parties P_i. Each party P_i has a secret x_i
- All parties jointly compute $y=f(x_1, \ldots, x_n)$
  - without revealing information on any secret x_i (except y)
- The computation must preserve certain security properties
  - Even if some parties collude and attack the protocol

- Basic solutions : rely on TTP
  - Each party sends her secret x_i to TTP;
  - TTP computes $y=f(x_1, \ldots, x_n)$ and sends it to a verifier
  - Verifier sends y to the parties (that may verify it too)
  - Eg the voting protocol with FHE (see section 1)
- Can we do as well without any TTP ?

Multi-party Computation without TTP

- Eg: compute Σx_i



- Note this scheme is not resistant facing corruption(s)

Oblivious transfer 1 among 2

- Alice has 2 plaintexts M0 and M1
- Bob asks Alice to send him M_s without revealing to Alice he wants M_0 or M_1.

Oblivious transfer 1 among 2

- Alice has 2 plaintexts M0 and M1
- Bob asks Alice to send him M_s without revealing to Alice he wants M_0 or M_1.

- One solution: (with multiplicative RSA)
  - Alice has RSA public (n,e) and secret d
  - Alice chooses random r_0 and r_1
    and she sends $x_0=r_0^e$ mod n and $x_1=r_1^e$ mod n to Bob
  - Bob chooses random k and sends $v=(x_s + k^e)$ mod n to Alice
  - Alice compute $C_0 = M_0+(v-x_0)^d$ mod n and $C_1 = M_1+(v-x_1)^d$ mod n
    She sends C_0 and C_1 to Bob
  - Bob computes $C_s – k$ and obtains his desired M_s.

- Note : a solution with FHE sends only one message C
  (but Alice computes all C_i with Bob public key)

Secret sharing problem « k among n »:

- S is a shared secret among n entities :
  - S is known by a TTP
  - S is represented by $D_1, \ldots, D_n$ with D_i secret of i
  - Knowledge of at least k values enables to compute S
  - Knowledge of less of k-1 D_i provides no information on S

Shamir protocol for secret sharing

- Use error correcting codes…
- Let F a (large) finite fiels such that S is uniquely and secretly represented in F
  - Prob(S=x) = 1/card(F)
- **Shamir's Proocol**
  - Let $f(X) = S + a_1.X + a_2.X^2 + \ldots + a_{k-1}X^{k-1}$ with $a_1, \ldots, a_k$ randomly chosen in F (let $a_0=S$)
  - Let n distinct elements $w_i \neq 0$ in F (for instance $w_i = i$ if characteristic(F) > n, or $w_i = g^i$ etc)
  - Each party i owns $(w_i, f(w_i))$
- **Multiparty computation of the secret by k parties :**
  - by interpolation of f (degree k-1) from k values f(w_i) : CRT
  - If less than k-1 values: then all values for S have same probability
- Moreover: resist to errors
  - possibility of correcting r errors (or attacks)
    - with k+r values si r ≥ 2.#errors

Shamir’s protocol properties

- **Perfect secrecy** (indistingability, like OTP)
- Minimal: la taille de chaque Di n’est pas plus grande que la taille de S
- **Dynamic** possible to change the ploynomial from time to time
- **Extendable** : adding paties is possible
- **Flexible**: party with high priority owns several values
- But requires confidence in the TTP that distributes the value

Conclusion Outsourcing computations and security

1. Computation with encrypted data : FHE
2. Interactive verification of results
3. Zero-knowledge proofs
   - Interactive zero-knowledge protocols
   - exercise
4. Secure multiparty Computations