

Summer school on Applied Cryptography  
Mykonos, September 27 – October 1, 2010

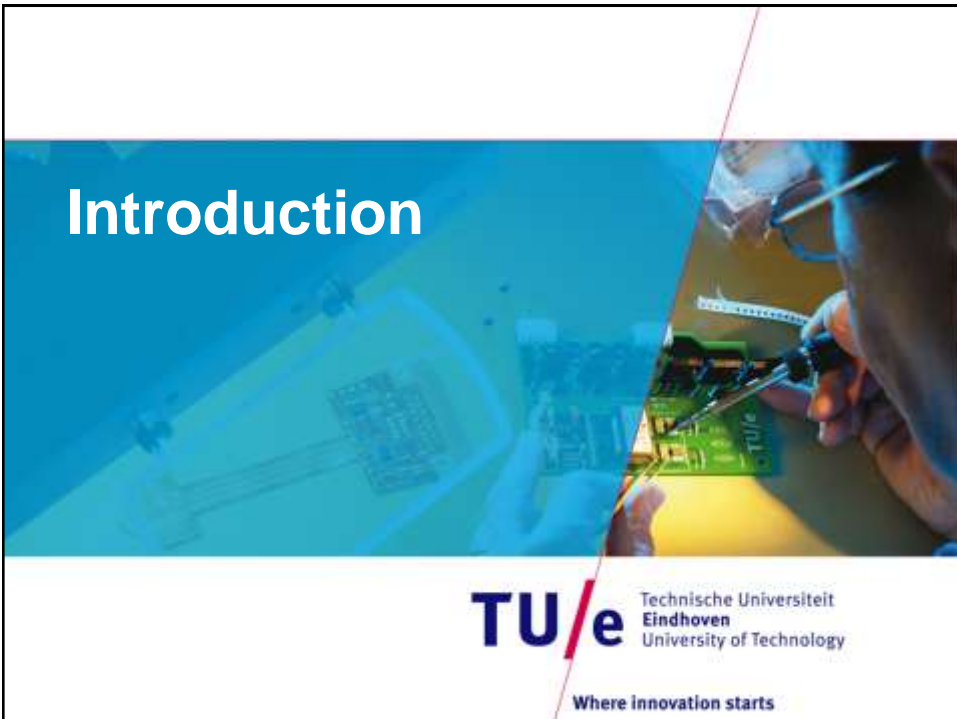
# Protocols for Multi-Party Computation

September 29, 2010

Berry Schoenmakers  
Coding & Crypto group  
Department of Mathematics & Computer Science

**TU/e** Technische Universiteit  
Eindhoven  
University of Technology

Where innovation starts

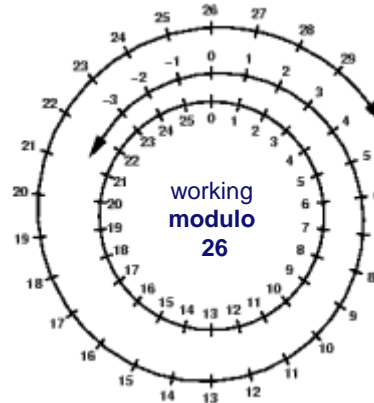


# Introduction

**TU/e** Technische Universiteit  
Eindhoven  
University of Technology

Where innovation starts

## Crypto 1.0 - Caesar cipher



- Rotate the alphabet by a fixed offset

## Crypto 1.0

### Crypto 1.0 concerns

- encryption and authentication of data
- during communication and storage/retrieval

### Crypto 1.0 primitives:

- **Symmetric (secret key)**
  - Stream/block ciphers
  - Message authentication codes
- **Asymmetric (public key)**
  - Public-key encryption
  - Digital signatures
  - Key-exchange protocols
- **Keyless**
  - Cryptographic hash functions

## Crypto 2.0

### Crypto 2.0 additionally concerns

- computing with encrypted data
- partial information release of data
- hiding identity of data owners or any link with them

### Crypto 2.0 primitives:

- **homomorphic encryption** Rivest, Adleman, Dertouzos '78
- **secret sharing** Blakley '79, Shamir '79
- **blind signatures** Chaum '82
- **oblivious transfer** M. Rabin. '81, EGL '85
- **zero-knowledge proofs** Goldwasser, Micali, Rackoff '85, GMW' 86
- **secure two/multi-party computation** Yao '82-86, GMW'87, BGW'88, CCD'88

## Crypto 2.0 vs Crypto 1.0

- **Crypto 1.0 encryption and authentication protects against malicious outsiders** (attacks on the storage or communication media)
  - Can be done using symmetric crypto
- **Crypto 2.0 primitives additionally protect against malicious insiders** (attacks by the other parties engaged in the protocols that you are running).
  - Needs more powerful cryptographic primitives, asymmetric crypto ... harder to do efficiently

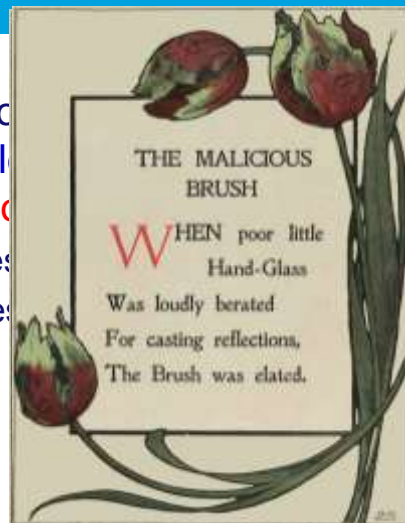
## Secure multiparty computation =

### NOT ABOUT:

- **Secure Computing:** computation within a tamper-resistant device (e.g., smart card), or on a private, standalone computer
- **Trusted Computing Initiative:** using a Trusted Platform Module (TPM) to provide a protected boot process
- **Defensive Computing:** secure (web)programming
- ... ?

## Secure multiparty computation =

- Performing joint cryptographic protocols between multiple parties where some of these parties may be malicious (in whatever way, for whatever reason)
- Cryptographic protocols between multiple parties may be malicious where some of these parties may be malicious
  - correctness: correct results
  - privacy: keeping parties' secrets



THE MALICIOUS BRUSH  
(from Fun & Nonsense by Willard Bonte)

## The Love Game

### Matching without Embarrassment

Alice and Bob want to find out if they're interested in each other:

Alice	Bob	Match?
No	No	-
No	Yes	-
Yes	No	-
Yes	Yes	♥



Privacy problem:

If Alice does **not** like Bob, she shouldn't know whether Bob does, **not** like Alice, he shouldn't know whether Alice is interested

If Alice is interested she learns if Bob is so too

## The Love Game – Reformulated

Two (or more parties) want to evaluate the logical AND function on secret inputs  $x$  and  $y$ :

$x$	$y$	$x \wedge y$
0	0	0
0	1	0
1	0	0
1	1	1

Note:  $x \wedge y = x y$   
(product)

Privacy requirement: protocol should not leak any information on inputs  $x$  and  $y$  **other than implied by the output value  $x \wedge y$**

## Any computable function from NAND

x	y	$\neg(x \wedge y)$
0	0	1
0	1	1
1	0	1
1	1	0

Theorem:  
NAND gate (a.k.a. “Sheffer stroke”) is functionally complete

Protocols for evaluating NAND gates securely  
IMPLY

protocols for evaluating any Turing computable function  $f$  securely

Recipe: construct an efficient circuit for  $f$  and  
devise protocols that work for arbitrary circuits

## “Love Game” for Alice and Bob

Alice’s rule:



separator:



Bob’s rule:



Suppose Alice thinks “yes”



Suppose Bob thinks “yes”



Alice and Bob make a random cut and open the deck ...

## Matching without embarrassments



Match!



These 3 cases  
are indistinguishable.

Hence if you put 'no'  
you don't find out the  
other's preference.

## Example applications

- **Secure Matching:** match **passenger list** of a US-bound flight with the **No Fly List**:

$$f(x, y) = x \cap y$$

- **Secure Elections, e.g.:**

$$f(x_1, x_2, \dots, x_n) = x_1 + x_2 + \dots + x_n$$

- **Secure Auctions, e.g.:**

$$f(x_1, x_2, \dots, x_n) = \max(x_1, x_2, \dots, x_n)$$

- **Secure Supply Chain Management**
  - E.g., optimization by linear programming

## Secure computation in practice

- E-voting, e.g., CyberVote, Helios
- Denmark, January 2008:
  - SCET/SIMAP projects (Univ. Aarhus)
  - **large-scale secure auction**
  - 1200 farmers traded sugar beet “production right quota”
- Double-auction with sealed bids:
  - avoids the use of a trusted auctioneer who would see all the farmer’s bids
  - secure integer comparison is used to rank the bids, without seeing the exact values of the bids

## Some EU projects on secure multiparty computation

- SPEED (FP6 EU)
  - **S**ignal **P**rocessing in the **E**ncrypt**E**d **D**omain
  - E.g., biometric face recognition under the encryption
- CACE (FP7 EU):
  - **C**omputer **A**ided **C**ryptography **E**ngineering
  - Incl. tools for zeroknowledge proofs and secure multiparty computation
- SecureSCM (FP7 EU):
  - **S**ecure **S**upply **C**hain **M**anagement
  - Companies along a supply chain want to reach a **global optimum**, but **without giving away their own (local) data**
    - Focus on Secure Linear Programming (Simplex algorithms)

## Languages and compilers for ZK and secure computation protocols

- Ad hoc, efficient implementation of advanced Crypto 2.0 primitives may be cumbersome and error-prone.
- Design of domain-specific programming languages together with efficient interpreters/compilers should make this technology generally accessible.
  - Thus, a high level of abstraction
- Lots of work in progress: Fairplay, VIFF, CACE, ...

## Virtual Ideal Functionality Framework

- VIFF 1.0 released December 14<sup>th</sup>, 2009. See [viff.dk](http://viff.dk) (Aarhus University)
- Free software. Runs on top of Python
- Implements version of secure **multi-party** computation

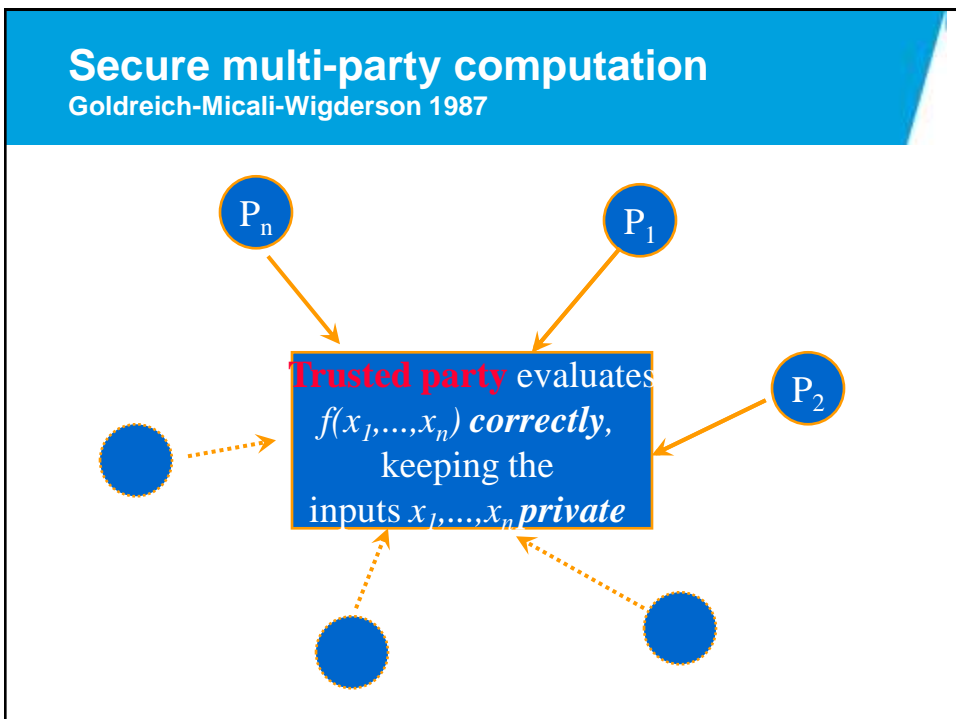
Fragment of a VIFF program:

```

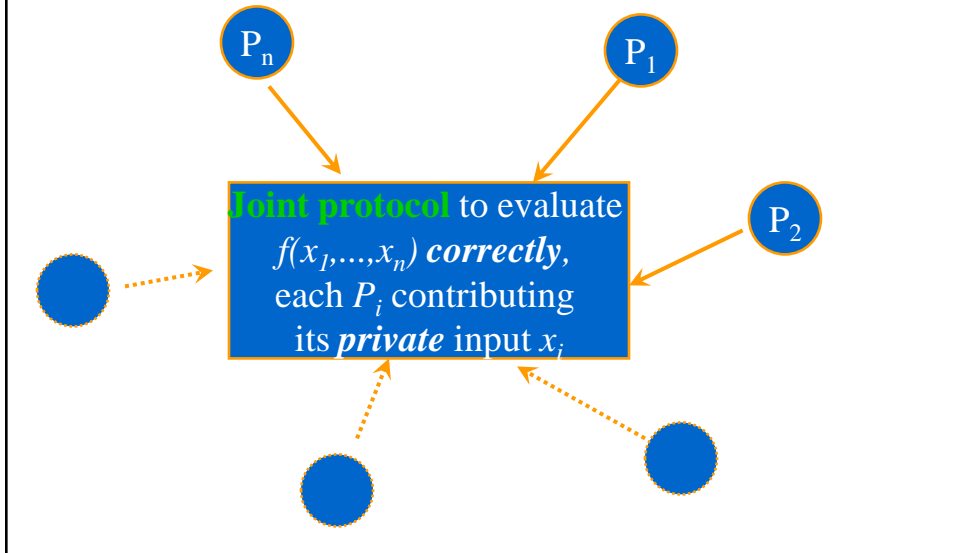
34
35 2p = GF(1031) // calculations are modulo p = 1031
36
37 id, players = load_config(args[0]) // the players that will share the value of x
38
39 def protocol(rt):
40
41     def got_result(result):
42         print "Product:", result
43         rt.shutdown()
44
45     x = rt.pvaw_share_random(2p) // x is assigned a random value
46     for _ in xrange(options.count): // x is squared a number of times
47         x = x * x
48     x = rt.open(x) // final value of x is revealed
49     x.addCallback(got_result)

```

Note: none of the **players** ever sees the intermediate values of x



## Eliminating the trusted party



## Ideal Functionality

- **Ideal SFE (Secure Function Evaluation):**
  - trusted party receives private inputs from  $P_1, \dots, P_n$
  - trusted party evaluates  $f$
  - and returns private outputs to  $P_1, \dots, P_n$ .
- **MPC vs. SFE:** MPC can be reactive, have state over time. SFE is like a one-shot thing.
- **“MPC vs. SFE : Unconditional and Computational Security” by Martin Hirt, Ueli Maurer, and Vassilis Zikas” ASIACRYPT 2008:**

“For the computational-security case, Ishai *et al.* [IKLP06] gave a *protocol* for SFE which tolerates an adversary who can either corrupt  $t_a < n/2$  players actively, or, alternatively,  $t_p < n$  players passively. They also showed that such an adversary cannot be tolerated for MPC.”

## Ideal Functionality / Real Protocol

- Any attack on the real protocol should also apply to the ideal functionality – no other attacks should be possible.
- Adversary  $A$  controls subset of parties.
  - Static vs. adaptive adversary
  - Computationally bounded vs. unbounded
  - Semi-honest vs. malicious (=passive vs. active)
    - also fail-stop (crashing) adversaries
- Simulation paradigm
  - $\forall A \exists S$  ideal and real indistinguishable

## Standalone vs. UC

- Standalone simulation:
  - Considers one instance of the protocol, which must be simulated such that an adversary cannot distinguish
  - $\forall A \exists S$  ideal and real indistinguishable
- Universally composable:
  - Includes a so-called **environment**  $Z$ , which can use the protocol in arbitrary constellations of protocol instances (incl. instances of the protocol itself)
  - $\forall A \exists S \forall Z$  ideal and real indistinguishable  
(or,  $\forall A \forall Z \exists S$  ideal and real indistinguishable)

## Alternative: Property based security

- **Privacy**
  - Shown by zeroknowledge-like reasoning (simulation)
- **Correctness**
  - Shown by soundness-like reasoning
- **Robustness**
  - Shown by completeness-like reasoning
    - Crashing parties should not get any advantage

## CIA

- **Confidentiality:**
  - Private inputs and private outputs
- **Integrity:**
  - Correctness of the outputs w.r.t. function and inputs
- **Availability:**
  - n-party ( $n > 2$ ) -> **robustness**
    - Honest parties get output, kicking out crashing parties
      - *Crashing should not give an advantage!*
  - 2-party -> **fairness**
    - Both parties get output, or both get nothing.
    - Impossibility result Cleve says that this is strictly speaking not possible. But relaxed ...

## Main general approaches

- (1978, Rivest et al. “Privacy homomorphisms”)
- **Early 1980s:**
  - Yao’s garbled circuits
- **Mid 1980s: VSS-based**
  - VSS = Verifiable Secret Sharing
  - Information-theoretic (unconditional) security
  - Needs private channels
- **Early 1990s: THC-based**
  - THC = Threshold Homomorphic Cryptosystems
  - Computational security (intractability)
  - ... fully homomorphic cryptosystems

## Communication Primitives (for voting)

- **Secure channel:** communication channel protected against eavesdropping, possibly providing end-to-end authentication as well, achieving **computational security**.
- **Private channel:** same as secure channel, but achieving **information-theoretic security**.
- **Untappable channel:** **totally unobservable** (out-of-band) communication channel.
- **Anonymous channel:** communication channel which **hides the sender’s identity**; possibly allows for an acknowledgement by the receiver as well.
- **Bulletin Board:** **publicly readable broadcast** channel, possibly with authenticated write operations.
- Also directed links, and combinations

## Oblivious Transfer (OT)

- OT is complete for MPC
- Rabin OT:
  - Sender holds a bit  $b$ , and receiver gets bit  $b$  with prob. 50%, and  $\perp$  otherwise
- 1-out-of-2 OT:
  - Reduces to Rabin OT (cost factor  $k$ , security parameter  $k$ )
  - Sender holds bits  $b_0$  and  $b_1$ , receiver inputs a selection bit  $s$ , and receives  $b_s$  (and no information on  $b_{1-s}$ ):

$$\text{OT}(b_0, b_1; s) = b_s$$

## Multiplication from OT

- Note that:  $\text{OT}(b_0, b_1; s) = (1-s) b_0 + s b_1 = b_s$
- Alice holds bit  $x$ , uses  $0, x$  as input to OT
- Bob holds bit  $y$ , uses  $y$  as input to OT

$$\text{OT}(0, x; y) = (1-y) 0 + y x = x y$$

- Bob gets  $x y$  from OT and sends it to Alice.

Protocol with **private inputs** and **public output**.

We need, in general, **hidden input** and **hidden output**  
where **hidden = shared or encrypted**

## General 2-party multiplication

- Alice and Bob hold shares of inputs and output.
- Shared input  $x = x_a + x_b$
- Shared input  $y = y_a + y_b$
- Goal: shared output  $z = z_a + z_b = x y$

$$x y = (x_a + x_b)(y_a + y_b) = x_a y_a + x_b y_a + x_a y_b + x_b y_b$$

- Put  $z_a = x_a y_a + x_b y_a$ , using  $OT(0, x_b; y_a) = x_b y_a$
- Put  $z_b = x_a y_b + x_b y_b$ , using  $OT(0, y_b; x_a) = y_b x_a$
- Security ? ...

## General 2-party multiplication

- Need some randomness
- Alice picks random bit  $r_a$
- Bob picks random bit  $r_b$
- Put  $z_a = x_a y_a + r_a + OT(r_b, x_b + r_b; y_a)$ 

$$= x_a y_a + r_a + r_b (1 - y_a) + (x_b + r_b) y_a$$

$$= x_a y_a + r_a + r_b + x_b y_a$$
- Similarly, put  $z_b = x_b y_b + r_a + r_b + x_a y_b$
- Mutual randomness  $r_a + r_b$  ensures that views of a corrupt party can be simulated.

## Arbitrary functions $f$

- Multiplication modulo 2
- Addition modulo 2 on shared inputs is easy:
  - Alice and Bob hold shares of inputs and output.
  - Shared input  $x = x_a + x_b$
  - Shared input  $y = y_a + y_b$
  - Put  $z_a = x_a + y_a$  ,  $z_b = x_b + y_b$
  - Shared output  $z = z_a + z_b = x + y$
- Given function  $f$ , construct a circuit over  $GF(2)$ , and evaluate the circuit gate by gate.



“Circuit thinking”

**TU/e** Technische Universiteit  
Eindhoven  
University of Technology

Where innovation starts



## “Circuit thinking”

Finding **optimal** circuits for given functions over a given base of gate-types

### Majority function

- Suppose we work over  $GF(2)$ .
- $\text{maj}(x,y,z) = xy + yz + zx$
- Best circuit for majority?
  - Lowest number of multiplications
- Three:  $xy, yz, zx$  ?
- Two:  $x(y+z), yz$  ?
- One:  $(x+y)(y+z) + y$  !

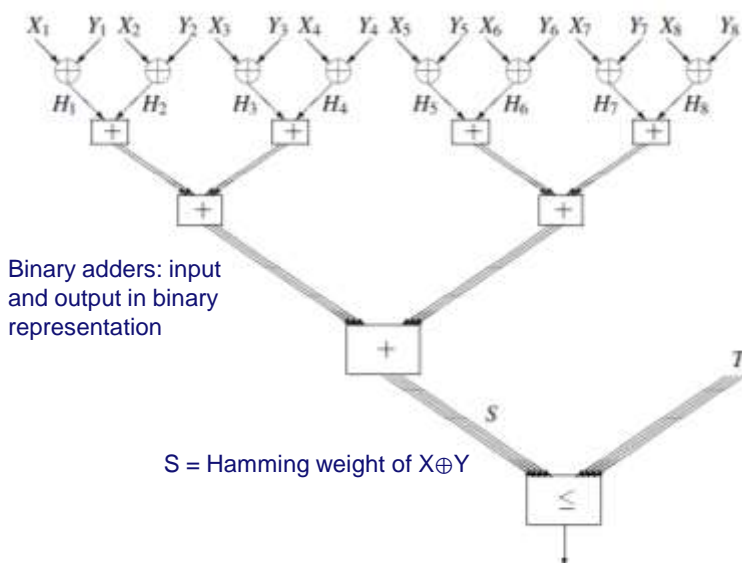
## Binary addition circuits

- **Input:**  $x_0, \dots, x_{m-1}, y_0, \dots, y_{m-1}$
- **Output:**  $z_0, \dots, z_m$  where  $z = x+y$
- **Optimal circuit:**  $c_{-1} = 0$ 

$$z_i = x_i + y_i + c_{i-1}$$

$$c_i = x_i y_i + x_i c_{i-1} + y_i c_{i-1} = \text{maj}(x_i, y_i, c_{i-1})$$
- Exactly  $m$  multiplications in total.

## Secure matcher: $d(x,y) \leq T$



## Circuit for Hamming weight

- **Input:** binary  $z_0, \dots, z_{m-1}$   
**Output:** binary  $h_0, \dots, h_{l-1}$  where  $h = \sum h_j 2^j = \sum z_i$
- **Straightforward approach:**
  - Assume  $m = 2^k$
  - Split input sequence into two halves
  - Compute Hamming weight for both parts
  - Use addition circuit to sum both values
    - Suppose addition costs  $k$  for  $k$ -bit inputs
- **Complexity as function of  $k$ :**

$$T_1 = 1$$

$$T_k = 2 T_{k-1} + k = 2^{k+1} - k - 2 = 2m + o(m)$$

## Optimal Circuit for Hamming weight

- **Input:** binary  $z_0, \dots, z_{m-1}$   
**Output:** binary  $h_0, \dots, h_{l-1}$  where  $h = \sum h_j 2^j = \sum z_i$
- **Optimal approach (Boyar-Peralta):**
  - Assume  $m = 2^k - 1$
  - Split input sequence into three parts:
    - length  $2^{k-1} - 1$  (two parts) and length 1
  - Compute Hamming weight for two long parts recursively
  - Use addition circuit to sum three values, **using single bit as carry input** at no extra cost.
- **Complexity:**

$$S_1 = 0$$

$$S_k = 2 S_{k-1} + k - 1 = T_{k-1} = m + o(m)$$
- **Generalizes to all  $m$ , hence saving a factor of 2.**

## Gates and circuits

A **circuit** is defined as a pure composition of gates, that is, obtained by connecting input/output wires of gates to form a network.

A **gate** is defined as any protocol which is not a circuit.

- Typically a gate requires **interaction** (usually a threshold decryption or share opening or OT step)

Examples:

- Addition, multiplication gates
- Random bit gate, sink gate
- Public input gate, constant gate, public output gate
- Least-significant bit gate
- Least-significant bits gate, Least-significant bits circuit
- Addition circuit, Subtraction circuit

## Optimal circuits

- Circuits have been studied extensively
  - Minimizing size (total number of gates)
  - Minimizing depth (length of critical path)
- For secure computation
  - Additions essentially for free
  - Multiplications dominate the cost
    - also for Yao with “Xor-for-free technique”
  - Especially important for use of fully homomorphic approach, if no (threshold) decryptions are done at any intermediate stage
- **Multiplicative complexity**: only counts multiplications.
  - Notion introduced early on, studied by Schnorr and others
  - We also like to limit the **multiplicative depth**

## Solution strategy

- Choose base of gates:
  - Finite field
  - Finite ring
- Find efficient circuits over this base
- Solutions are secure by construction
  
- Try a direct approach (includes use of circuits):
  - Specific protocol, performing a threshold decryption or opening along the way.

## Lower bounds

- Multiplicative complexity of  $x \cdot y$  for  $m$ -bit values
  - $m$  for fields (and rings) of characteristic 2
  - $2m - 1$  otherwise (see later)
  
- Majority function:  $ab+ac+bc$
  
- In **one** multiplication
  - over  $\text{GF}(2)$
  - over  $\text{GF}(q)$  of characteristic 2
- But otherwise, **two** multiplications
  - over  $\text{GF}(q)$  of characteristic  $\neq 2$
  - over rings of characteristic  $\neq 2$ , like  $\mathbb{Z}_N$ ,  $N=p \cdot q$

## Hence, a choice ...

- **Choosing characteristic 2 may be advantageous**
  - Use Goldwasser-Micali encryption (quadratic residues) to get xor-homomorphic property
- Fully homomorphic lattice-based cryptosystems can be taken with an even modulus, using the least significant bit of the plaintext should then give characteristic 2.
- When using VSS, consider use of  $GF(q)$  with  $q=2^r$
- Open problem: Discrete Log based characteristic 2 homomorphic cryptosystem ?

## THC-based examples

# THC

Tetra

Hydro

Cannabinol

- =
- Public Key (
- + **Homomorphic** Encryption
- + **Threshold** Decryption



## Homomorphic encryption

- Message  $x$ , public key  $pk$
- Probabilistic public key encryption, e.g. ElGamal:

$$E_{pk}[x, r] = (g^r, pk^r g^x), \quad \text{randomness } r$$

- Homomorphic property:

$$E_{pk}[x, r] * E_{pk}[y, s] = E_{pk}[x + y, r + s]$$

- In shorthand notation:

$$E[x] * E[y] = E[x + y]$$

$$E[x] / E[y] = E[x - y]$$

$$E[x]^c = E[c \cdot x] \quad (E[x] \text{ multiplied } c \text{ times with itself})$$

## Popular choice of THCs

- |  |  |
|--|--|
| <ul style="list-style-type: none"> <li>• Homomorphic ElGamal<br/><math>E_{g,h}(m,r) = (g^r, h^r g^m)</math></li> <li>• DDH assumption</li> <li>• Pros:             <ul style="list-style-type: none"> <li>- <i>efficient DKG</i> to share private key <math>\alpha = \log_g h</math> [Ped91, ..., AF04]</li> <li>- allows for <i>elliptic curves</i> (<i>exponential security</i>)</li> </ul> </li> <li>• Cons:             <ul style="list-style-type: none"> <li>- <i>limited decryption</i> (only full decryption of <math>g^m</math>, from which <math>m</math> needs to be recovered still).</li> </ul> </li> </ul> | <ul style="list-style-type: none"> <li>• Paillier<br/><math>E_n(m,r) = (1+n)^m r^n \bmod n^2</math></li> <li>• RSA-like assumption</li> <li>• Pros:             <ul style="list-style-type: none"> <li>- <i>full decryption</i> of message <math>m</math></li> </ul> </li> <li>• Cons:             <ul style="list-style-type: none"> <li>- <i>expensive DKG</i> for generating a shared RSA modulus [Gil99, ACS02]. Cost of DKG may dominate total cost.</li> <li>- only subexponential security</li> </ul> </li> </ul> |
|--|--|

## Popular choice of THCs (cont.)

- ELGamal-Paillier amalgam (CraSho'02, DamJur'03)

$$E_{g,h,n}(m,r) = (g^s \bmod n, (1+n)^m (h^s \bmod n)^n \bmod n^2)$$

- DDH and RSA-like assumption
- Pros:
  - *full decryption* of message  $m$
  - *expensive DKG* now only at system setup (single, system-wide RSA modulus  $n$  for all users)
- Cons:
  - large overhead due to large ciphertexts, e.g. compared to ElGamal combined with elliptic curves
    - even if secure computation is mostly bitwise (Boolean circuits)
  - relies on two assumptions:
    - factorization of RSA modulus  $n$  is actually a trapdoor

## Threshold decryption

Let  $sk$  denote the private key for  $pk$

- **Standard:**  $sk$  held by one party
- **Threshold:**  $sk$  shared between multiple parties
  
- **Decryption only succeeds if a majority of the parties cooperates to decrypt a given ciphertext**
  - **central idea:** during protocol runs only decrypt “random”, yet useful messages

## Examples

- **Given  $x, y$ :**
  1. **Equality test:**  $x = y$
  2. **Multiplication:**  $xy$  with  $x$  two-valued
  3. **LSB – least significant bit:**  $x_0$
  4. **Integer comparison:**  $x > y$  (“Yao’s millionaires”)
  
- **For simplicity:** **two, semi-honest parties**

## Equality test $x = y$

- Input:  $E[x], E[y]$
- Output:  $x = y$  (public output)
- Protocol:
  - Publicly compute  $E[x]/E[y] = E[x-y]$
  - Party 1 sets  $E[x-y]^{r_1} = E[r_1(x-y)]$  random  $r_1 \neq 0$
  - Party 2 sets  $E[r_1(x-y)]^{r_2} = E[r_1 r_2(x-y)]$  random  $r_2 \neq 0$
  - Threshold decrypt  $E[r_1 r_2(x-y)] \rightarrow g^z = g^{(r_1 r_2(x-y))}$
  - Now  $x=y$  if and only if  $g^z=1$ .

Uniform random non-zero value if  $x \neq y$

## Multiplication $xy$

- Input:  $E[x], E[y]$ , with  $x \in \{1, -1\}$
- Output:  $E[xy]$
- Protocol:
  - Party 1 picks random  $s_1 \in \{1, -1\}$ , and sets:  
 $E[x]^{s_1} = E[s_1 x]$ ,  $E[y]^{s_1} = E[s_1 y]$
  - Party 2 picks random  $s_2 \in \{1, -1\}$ , and sets:  
 $E[s_1 x]^{s_2} = E[s_1 s_2 x]$ ,  $E[s_1 y]^{s_2} = E[s_1 s_2 y]$
  - Threshold decrypt  $E[s_1 s_2 x] \rightarrow z = s_1 s_2 x$
  - Publicly compute, using  $s_1^2 = s_2^2 = 1$ :  
 $E[s_1 s_2 y]^z = E[s_1 s_2 y s_1 s_2 x] = E[xy]$

Uniform random value in  $\{1, -1\}$ : does not reveal any information on  $x$

## CDN01 framework

- THC-based approach
- Simulation of entire protocol for evaluating a circuit for  $f$  ultimately boils down to simulating each of the gates.
- Two options:
- Computationally indistinguishable simulation of multiplication gate:
  - Given  $E[x]$ ,  $E[y]$ , put **dummy encryption  $E[0]$**  for output
  - Leads to **non-tight reduction**, nightmare hybrid argument
- Statistical simulation of multiplication gate:
  - Given  $E[x]$ ,  $E[y]$  and  **$E[xy]$**  simulate the protocol.
  - Leads to **tight reduction**, and simple argument

## Simulation

- Given  $E[x]$ ,  $E[y]$ ,  $E[xy]$ , we want to simulate view of corrupted party, say  $P_1$ .
- $P_1$  sets  $E[x]^{s_1} = E[s_1x]$ ,  $E[y]^{s_1} = E[s_1y]$
- Simulator picks  $s'_2 \in \{1, -1\}$  random and sends  $E[s'_2]$  and  $E[xy]^{s'_2}$
- Simulate threshold decryption of  $E[s'_2]$  revealing  $s'_2$ 
  - **Using knowledge of the plaintext  $s'_2$**
- Note that, with  $s_2 := s'_2 s_1x$ , we have as required:
  - $E[s'_2] = E[s_1x]^{s_2}$
  - $E[xy]^{s'_2} = E[x y s'_2 s_1x] = E[s_1y]^{s_2}$

## Binary conversion gate Eurocrypt 2006

- Protocol for computing binary representation (bits) for a given integer value:

$$E(x) \rightarrow E(x_0), \dots, E(x_{m-1})$$

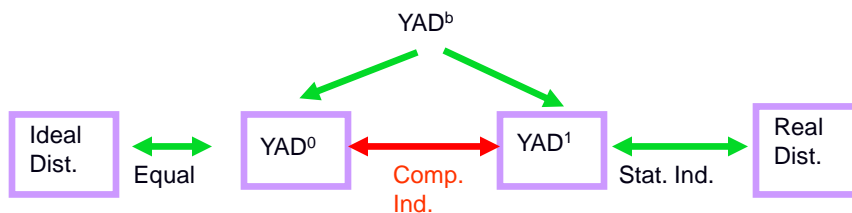
- Essential to use Paillier THC
  - in general, one cannot use ElGamal THC
    - input:  $E(x) = (g^r, h^r g^x)$
    - output:  $E(x_0) = (g^r, h^r g^{x_0})$ ,  $x_0$  = least significant bit of  $x$
    - but  $x_0$  is a hard-core bit  $\rightarrow$  would imply computing DLs (for prime order subgroup of  $Z_p^*$  and general cyclic groups)

## Protocol for LSB gate: $E(x) \rightarrow E(x_0)$

- Random bit gate  $E(r_0)$   $0 \leq r_0 < 2$
- Random value gate  $E(r_*)$   $0 \leq r_* < 2^k$
- Threshold-decrypt  $E(x + r_0 + 2r_*)$  to get
 
$$y = x + r_0 + 2r_*$$
- Output  $E(x_0) = E(y_0) E(r_0)^{1-2y_0}$
- Note:  $y_0 = x_0 \oplus r_0$  so  $x_0 = y_0 \oplus r_0 = y_0 + r_0 - 2y_0 r_0$

## Security proof

- If simulator gets  $E[x_0], \dots, E[x_1]$  as input, we cannot simulate statistically, as we don't have  $E[x_0 x_1]$  etc.
- To use the CDN-framework, suffices to simulate given  $x$  and  $x'$  (in the clear) and given  $E[b]$ , for the input  $E[\underline{x}]$ , where  $\underline{x} = x b + x' (1-b)$ .
- $E[\underline{x}]$  is computed from  $E[b]$ ,  $x$ ,  $x'$ .
- We can simulate LSB protocol by making consistent assignments for  $x$  as well  $x'$



## Protocol for $E[x] \rightarrow E[x \bmod L]$

- Given  $E[x]$
- Jointly generate random  $E[r]$  with  $0 \leq r < L$ 
  - Bitwise  $E[r_0], \dots, E[r_{n-1}]$
- Jointly generate random  $E[s] = E[\sum_i s_i]$
- Threshold decrypt  $E[x-r+Ls]$
- Compute  $y = (x-r+Ls) \bmod L = (x+r) \bmod L$
- Comparison:  $E[c] = E[L-1-y < r]$
- Correction:  $E[y+r]/E[c]^L = E[x \bmod L]$

## Integer comparison $x > y$

- Input:  $E[x]$ ,  $E[y]$
- Output:  $x > y$  (public output)
- 1<sup>st</sup> attempt (like equality test):
  - form  $E[x-y]$
  - multiply with random “positive”  $r$  to form  $E[r(x-y)]$
  - threshold decrypt to get  $r(x-y)$
  - decide  $x > y$  based on “sign” of  $r(x-y)$
- ... problem: **non-uniform value for  $r(x-y)$**

## Integer comparison $x > y$

- Resort to bit-by-bit methods:  $x = (x_{m-1}, \dots, x_0)_2$ ,  $y = (y_{m-1}, \dots, y_0)_2$
- Input:  $E[x_0], \dots, E[x_{m-1}]$ ,  $E[y_0], \dots, E[y_{m-1}]$   
Output:  $E[x > y]$  (encrypted bit equal to 1 iff  $x > y$ )
- Intuitively: compare most significant bits, until 1st difference found.
- But one must hide where the difference is found!!!
  - Use a **circuit** (oblivious program): data-independent execution path
  - **Central goal: find efficient circuits**
    - Ignore addition (for free due to homomorphic property)
    - **Minimize # of multiplication gates** **computational complexity**
    - **Minimize depth of circuit** (longest critical path) **round complexity**

Circuits for  $x > y$ 

(1/3)

- **Counterintuitive msb-to-lsb traversal** beats lsb-to-msb traversal
- **lsb-to-msb circuit:**
  - traverse  $x$  and  $y$  starting at **least significant bit**
  - if difference found, record whether  $x_i > y_i$  (i.e.,  $x_i = 1$  and  $y_i = 0$ )
  - continue all the way to the end, keeping last recorded result

$$t_0 = 0, \quad t_{i+1} = (1 - (x_i \oplus y_i)) t_i + x_i (1 - y_i)$$

$$= (1 - x_i - y_i + 2 x_i y_i) t_i + x_i - x_i y_i \quad \text{output: } t_m$$

- Additions/subtractions for free (homomorphic property)
- Per iteration only two multiplications:  $x_i y_i$  and  $(1 - x_i - y_i + 2 x_i y_i) t_i$

Circuits for  $x > y$ 

(2/3)

- **Recursive approach:** let  $x = X_1 X_0$ ,  $y = Y_1 Y_0$ ,  $|X_1| = |Y_1|$

x:	$X_1$	$X_0$
y:	$Y_1$	$Y_0$

Then:

$$x > y \Leftrightarrow X_1 > Y_1 \vee (X_1 = Y_1 \wedge X_0 > Y_0)$$

- **Three ways to split  $x$  and  $y$ :**
  - Split off leftmost element: **msb-to-lsb**
  - Split off rightmost element: **lsb-to-msb**
  - Split midway: **"divide & conquer"**

Circuits for  $x > y$ 

(3/3)

	# multiplications	circuit depth
msb-to-lsb $ X_1  =  Y_1  = 1$	$3m$	$m$
lsb-to-msb $ X_0  =  Y_0  = 1$	$2m$	$m$
divide and conquer $ X_1  =  Y_1  \approx  X_0  =  Y_0 $	$3m$	$\log_2 m$
theoretical	$O(m)$	$O(1)$

## Conclusions

- Efficient solutions available for specific problems such as voting, auctions, ...
- Secure multiparty computation (continues to) ask(s) for new ideas and techniques:
  - Different effects (e.g., obliviousness)
  - Different cost measures (e.g., addition for free, multiplications much more expensive)
- More experiments upcoming with implemented solutions to evaluate performance

## Conclusions 2.0

- **Crypto 2.0** started more than 30 years ago
- **Crypto 2.0** started being put into practice some 20 years ago
- **Efficient Crypto 2.0** solutions available for specific problems such as electronic cash, voting, auctions, ...
- **Crypto 2.0** research is blossoming and work on practical deployment is increasing steadily
- **Orthogonal** to Crypto 1.0 vs 2.0 there are many developments as well: e.g., leakage resilient crypto, quantum crypto, provable security, universal composability, ...

## Author's address

**Berry Schoenmakers**

**Coding and Crypto group  
Dept. of Mathematics and Computer Science  
Technical University of Eindhoven  
P.O. Box 513  
5600 MB Eindhoven  
The Netherlands**

**berry@win.tue.nl  
l.a.m.schoenmakers@tue.nl**

**<http://www.win.tue.nl/~berry/>**