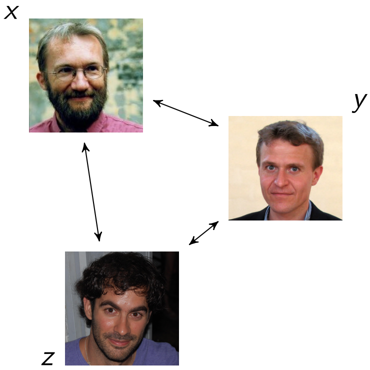
MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer

Tore Frederiksen *Marcel Keller*
Emmanuela Orsini Peter Scholl

Aarhus University
University of Bristol

31 May 2016

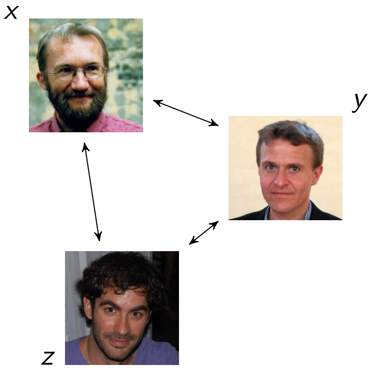
Secure Multiparty Computation



Wanted: $f(x, y, z)$

- Computation on secret inputs
- Replace trusted third party

Secure Multiparty Computation



Wanted: $f(x, y, z)$

- Computation on secret inputs
- Replace trusted third party
- Formulate f as circuit
- Central questions in MPC
  - How many trusted parties?
  - What deviation?

Multiparty Computation in This Talk

Security model

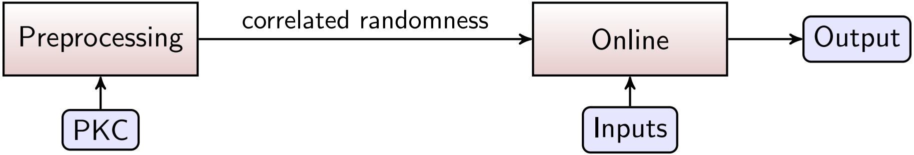
How many parties are how corrupted? In this work:

- Malicious adversary: Corrupted parties deviate from protocol.
- Dishonest majority of corrupted parties
  - Impossible without computational assumptions (PK crypto)
  - Shamir secret sharing does not help
  - No guaranteed termination

What Tools Do We Need?

- Linear secret sharing to store intermediate results
- Homomorphic authentication for active security
  $\Rightarrow$ "Free" linear computation!
- Multiplication is harder. We need public-key crypto.
  - Using this on intermediate values is hard.
    - How to assure correct behaviour?
    - How to avoid leakage if protocol fails?
  - Easier: Preprocess correlated randomness

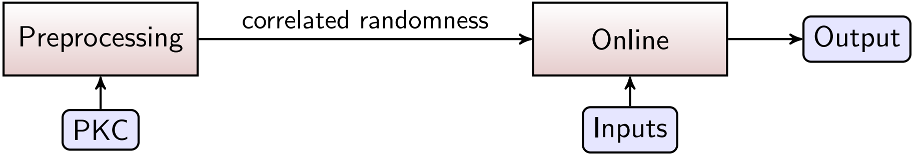
Malicious Offline-Online MPC Protocols



Advantages

- No secret inputs on the line when using crypto
  $\Rightarrow$ No one gets hurt if protocol aborts!
- Online computation might have many rounds,
  but preprocessing is constant-round.

Malicious Offline-Online MPC Protocols



SPDZ [Damgård, Pastro, Smart, Zakarias 2012]

Circuit over $\mathbb{F}_p$ (prime) or $\mathbb{F}_{2^k}$, preprocessing using somewhat homomorphic encryption

TinyOT [Nielsen, Nordholt, Orlandi, Burra 2012]

Circuit over $\mathbb{F}_2$, preprocessing using oblivious transfer

How to Share a Secret with Authentication

	Shares	MAC shares	MAC key
	x_1	$\gamma(x)_1$	α_1
	x_2	$\gamma(x)_2$	α_2
	x_3	$\gamma(x)_3$	α_3
	$x = \sum_i x_i$	$\alpha \cdot x = \sum_i \gamma(x)_i$	$\alpha = \sum_i \alpha_i$
	$= x$		

How to Share a Secret with Authentication

	Shares	MAC shares	MAC key
	$x_1 + y_1$	$\gamma(x)_1 + \gamma(y)_1$	α_1
	$x_2 + y_2$	$\gamma(x)_2 + \gamma(y)_2$	α_2
	$x_3 + y_3$	$\gamma(x)_3 + \gamma(y)_3$	α_3
	$x + y$ $= \sum_i x_i + y_i$	$\alpha \cdot (x + y)$ $= \sum_i \gamma(x)_i + \gamma(y)_i$	α $= \sum_i \alpha_i$
	$=$	$x + y$	

Multiplication with Random Triple (Beaver Randomization)

$$\begin{aligned} x \cdot y &= (x + a - a) \cdot (y + b - b) \\ &= (x + a) \cdot (y + b) - (y + b) \cdot a - (x + a) \cdot b + a \cdot b \end{aligned}$$

Multiplication with Random Triple (Beaver Randomization)

$$\boxed{x \cdot y} = (x + a - a) \cdot (y + b - b)$$

$$= \underbrace{(x + a)} \cdot \underbrace{(y + b)} - \underbrace{(y + b)} \cdot \boxed{a} - \underbrace{(x + a)} \cdot \boxed{b} + \boxed{a \cdot b}$$

Masked and opened

Random secret triple

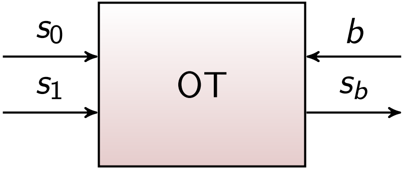
Preprocessing — Triple Production

Multiplication of secret values

- Somewhat homomorphic encryption (SPDZ)
  - Relatively expensive computation
  - Zero-knowledge proofs of correct ciphertext generation
- Oblivious transfer
  - Cheap computation with OT extension
  - Need to mitigate selective failure
- No multiplicative secret sharing for dishonest majority

1-out-of-2 Oblivious Transfer

Sender

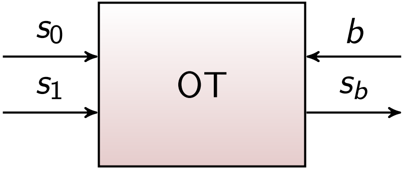


Receiver

- The **Sender** inputs two strings s_0 and s_1 and learns nothing.
- The **Receiver** inputs a bit b and learns only s_b.

1-out-of-2 Oblivious Transfer

Sender



Receiver

Assume s_0, s_1 represent elements in $\mathbb{F}$, and define $a = s_1 - s_0$:

$$
\begin{aligned}
s_b - s_0 &= (1 - b) \cdot s_0 + b \cdot s_1 - s_0 \\
&= b \cdot (s_1 - s_0) \\
&= b \cdot a
\end{aligned}
$$

OT Multiplication for Field $\mathbb{F}$

Passive security

Break down $\mathbb{F} \times \mathbb{F}$ multiplication to $\log |\mathbb{F}|$ multiplications of bit and element in $\mathbb{F}$ (previous slide):

$$x = \sum_{i=0}^{\log |\mathbb{F}|} 2^i \cdot x_i \quad \Rightarrow \quad x \cdot y = \sum_{0}^{\log |\mathbb{F}|} 2^i \cdot (x_i \cdot y)$$

Selective failure

Parties need to input the same value in several OT instances.
If not, a protocol might fail later depending on secret bits.
Non-failure reveals secret information!

Triple Generation

1. Parties sample random shares of a, b and the MAC key α
2. For additive sharings of $a \cdot b$, $a \cdot \alpha$, $b \cdot \alpha$
   - Every pair uses OT for secret sharing of product of two shares.
   - Compute product of two local shares and sum up.
3. Repeat for additive sharing of $a \cdot b \cdot \alpha$

Active Security

Need to mitigate selective failure attack:

- Check by opening some randomness ("sacrificing" some triples)
- Privacy amplification to dilute information that is revealed if check passes

Secure Triple Generation with OT

Binary circuits, $\mathbb{F} = \mathbb{F}_2$

- Generate enough triples
- Check some triples with cut-and-choose
- Recombine random subsets of the rest to remove leakage
- $9\times$ overhead over passive triple with MAC generation

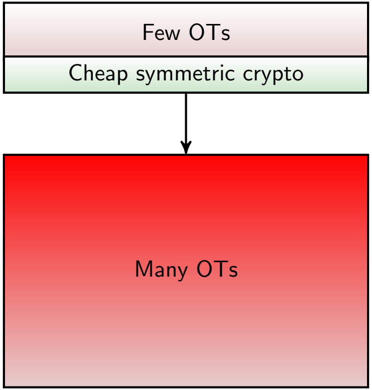
Arithmetic circuits for $\mathbb{F}$ large enough

- Hard enough to guess a random element of $\mathbb{F}$
- It suffices to randomly combine and check a few triples
- $3\times$ overhead over passive triple with MAC generation

Oblivious Transfer Implementation

- Plain OT: 10'000 per second (Chou and Orlandi)
- OT extension: 7 million per second on a 1 Gbit/s link
  `https://github.com/bristolcrypto/apricot`
- Cost of active security is negligible
- Essential cost is sending k bits per random OT
  for computational security k

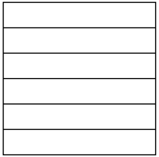
OT Extension — Basic Idea



go fast

go slow

OT Extension with Passive Security



1. Base OTs	k random OTs / k bits
2. Extend length with PRG	k random OTs / n bits
3. Introduce correlation	k correlated OTs / n bits
4. Transpose	n correlated OTs / k bits
5. Hash to break correlation	n random OTs / k bits

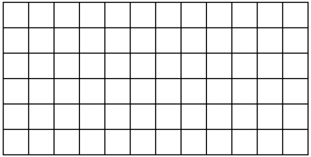
Computational security parameter	$k = 128$
Number of OTs produced	$n \geq 128$

OT Extension with Passive Security

1.	Base OTs	k random OTs / k bits
2.	Extend length with PRG	k random OTs / n bits
3.	Introduce correlation	k correlated OTs / n bits
4.	Transpose	n correlated OTs / k bits
5.	Hash to break correlation	n random OTs / k bits

Computational security parameter	$k = 128$
Number of OTs produced	$n \geq 128$

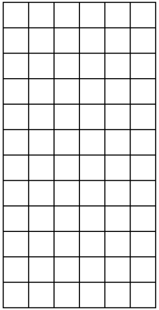
OT Extension with Passive Security



1. Base OTs	k random OTs / k bits
2. Extend length with PRG	k random OTs / n bits
3. Introduce correlation	k correlated OTs / n bits
4. Transpose	n correlated OTs / k bits
5. Hash to break correlation	n random OTs / k bits

Computational security parameter	$k = 128$
Number of OTs produced	$n \geq 128$

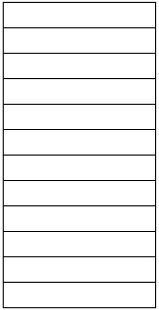
OT Extension with Passive Security



1. Base OTs	k random OTs / k bits
2. Extend length with PRG	k random OTs / n bits
3. Introduce correlation	k correlated OTs / n bits
4. Transpose	n correlated OTs / k bits
5. Hash to break correlation	n random OTs / k bits

Computational security parameter	$k = 128$
Number of OTs produced	$n \geq 128$

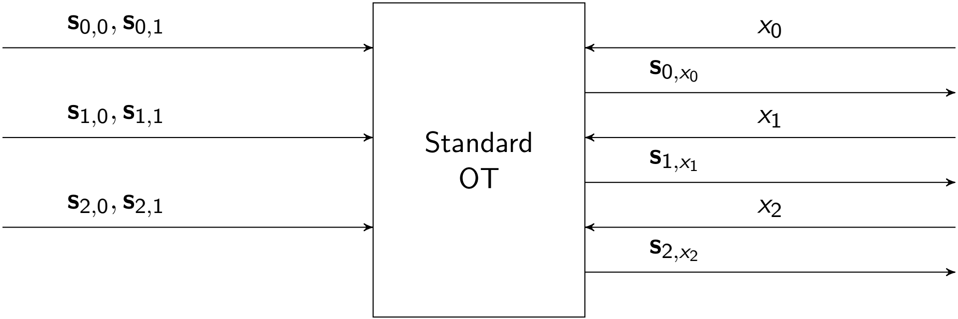
OT Extension with Passive Security



1. Base OTs	k random OTs / k bits
2. Extend length with PRG	k random OTs / n bits
3. Introduce correlation	k correlated OTs / n bits
4. Transpose	n correlated OTs / k bits
5. Hash to break correlation	n random OTs / k bits

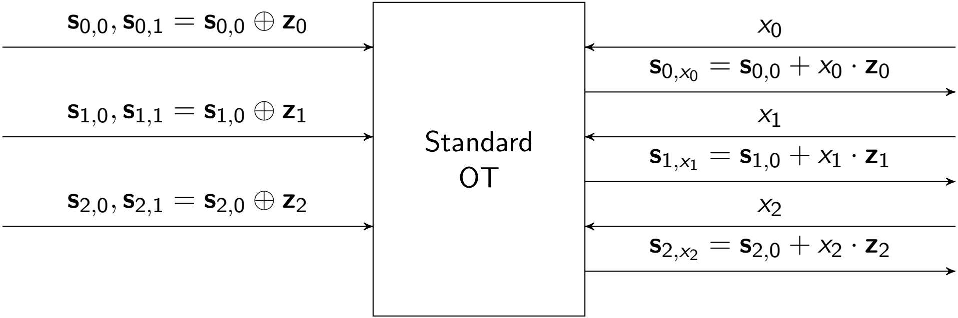
Computational security parameter	$k = 128$
Number of OTs produced	$n \geq 128$

Another Look at OT



x_i: selection bit
$\mathbf{s}_{i,0}, \mathbf{s}_{i,1}, \mathbf{t}_i, \mathbf{z}_i, \mathbf{y}$: strings

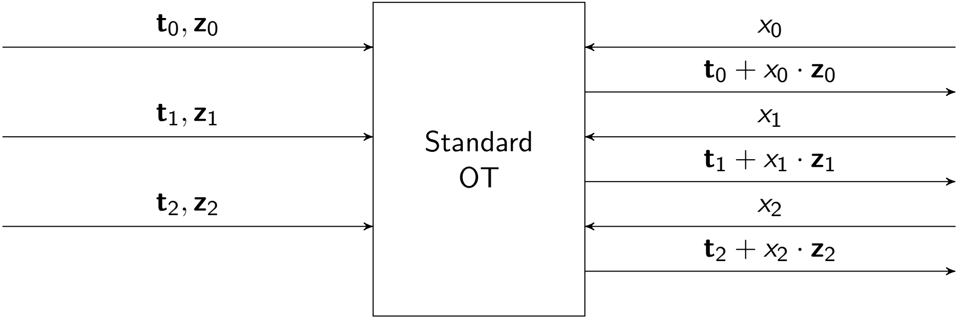
Another Look at OT



x_i: selection bit

$\mathbf{s}_{i,0}, \mathbf{s}_{i,1}, \mathbf{t}_i, \mathbf{z}_i, \mathbf{y}$: strings

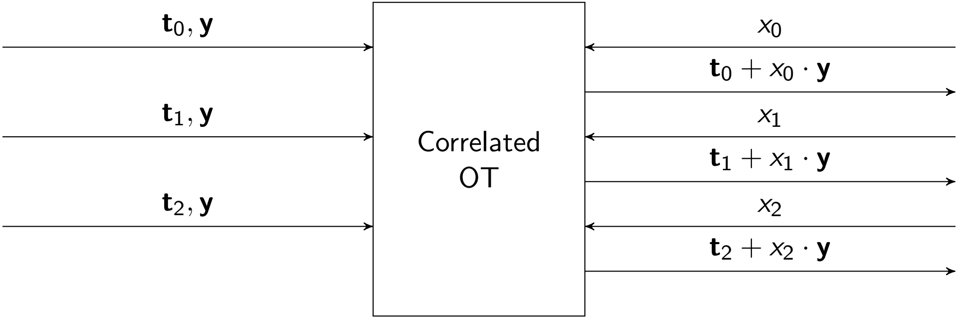
Another Look at OT



x_i: selection bit

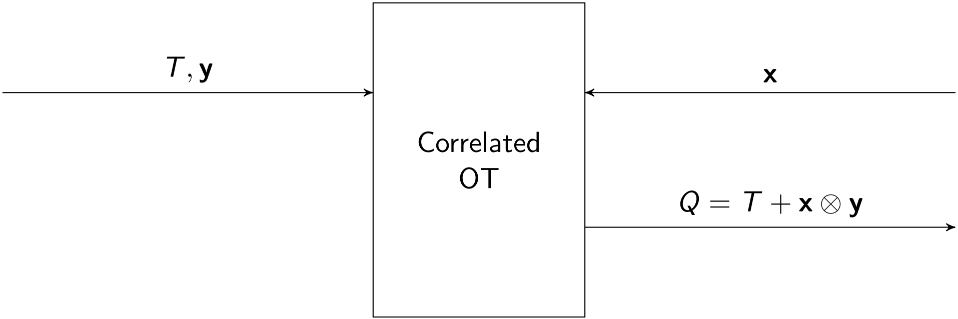
$\mathbf{s}_{i,0}, \mathbf{s}_{i,1}, \mathbf{t}_i, \mathbf{z}_i, \mathbf{y}$: strings

Another Look at OT



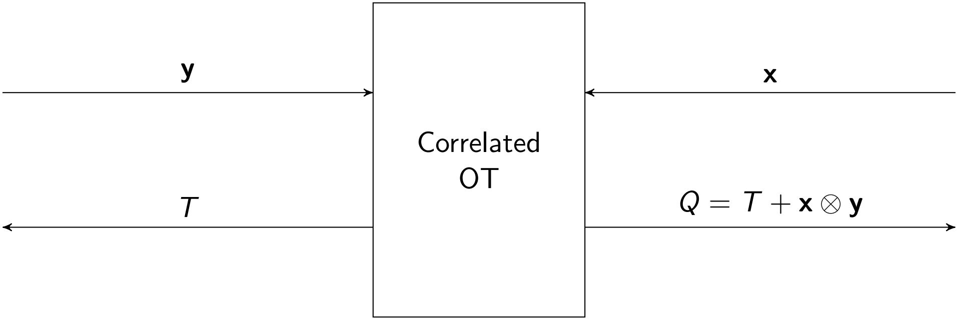
x_i: selection bit
$\mathbf{s}_{i,0}, \mathbf{s}_{i,1}, \mathbf{t}_i, \mathbf{z}_i, \mathbf{y}$: strings

Another Look at OT



$\mathbf{x}, \mathbf{y}$:	strings / vectors in $(\mathbb{F}_2)^k$ and $(\mathbb{F}_2)^n$, respectively
Q, T, Z:	matrices in $(\mathbb{F}_2)^{k \times n}$
$\mathbf{x} \otimes \mathbf{y}$:	tensor product, matrix of all possible products

Another Look at OT



$\mathbf{x}, \mathbf{y}$: strings / vectors in $(\mathbb{F}_2)^k$ and $(\mathbb{F}_2)^n$, respectively
Q, T, Z: matrices in $(\mathbb{F}_2)^{k \times n}$
$\mathbf{x} \otimes \mathbf{y}$: tensor product, matrix of all possible products

OT Extension with Active Security

Problem

- Party responsible for correlation (sender of base OT) can deviate
- $Q = T + \mathbf{x} \otimes \mathbf{y}$ not guaranteed

Solution

- Columns of $\mathbf{x} \otimes \mathbf{y}$: $(y_1 \cdot \mathbf{x}, \ldots, y_n \cdot \mathbf{x})$
- Base OT sender knows T and $\mathbf{y}$
- Sends random linear combination of columns in T and elements in $\mathbf{y}$ over the extension field $\mathbb{F}_{2^k}$

Software Implementation

If you have AES in the processor...

AES-based Cryptography

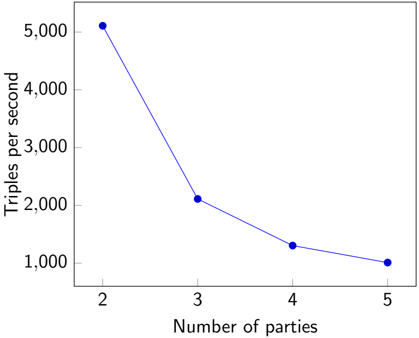
Pseudorandom generator

- $\text{PRG}(K) = \text{AES}_K(0), \text{AES}_K(1), \text{AES}_K(2), \ldots$
- Need to compute key schedule only once

Hashing

- $H(x) = \text{AES}_0(x) \oplus x$
- Simplified version of Matyas–Meyer–Oseas
- Input length is limited to 128 bits
- Unlike $H(x) = \text{AES}_x(0) \oplus x$ (Davies–Meyer), the key schedule is always the same.

Results – Triple Generation for 128-bit Fields



- $\mathbb{F}_{2^{128}}$ or $\mathbb{F}_p$ for 128-bit p
- Computational security 128
- Statistical security 64 (128 would cost $< 20\%$)
- 1 Gbit/s link
- 180'224 bits per triple (max. 5549 triples/s for 2)
- SPDZ: 369 or 24 triples/s ($\mathbb{F}_p$, covert or active)

100-Party Computation Goes Live!

Triple generation

	Triples/s	Triples/$/party
2 parties	45478	2.6e8
100 parties	242	1.0e6

100-party Vickrey second-price auction

	Time	Cost per party
t2.nano	9.0 s	$0.000017
c4.8xlarge	1.4 s	$0.000741

Triples cost 18 seconds or $0.0044 per party.

Conclusion

For n parties and security k, overall communication per triple:

- $\Omega(n(n-1)k \log|\mathbb{F}|)$ for all protocols in this line of work
- MASCOT: $\leq 13(n(n-1) \max(\log|\mathbb{F}|, k) \log|\mathbb{F}|)$ bits. Computation insignificant
- Open question: Asymptotic improvement?