KU LEUVEN

Introducing TopGear: an efficient and secure Zero-Knowledge proof for multiparty computation

Carsten Baum[3], Daniele Cozzo[1] & Nigel P. Smart[1,2]

[1]KU Leuven
ESAT/imec-COSIC

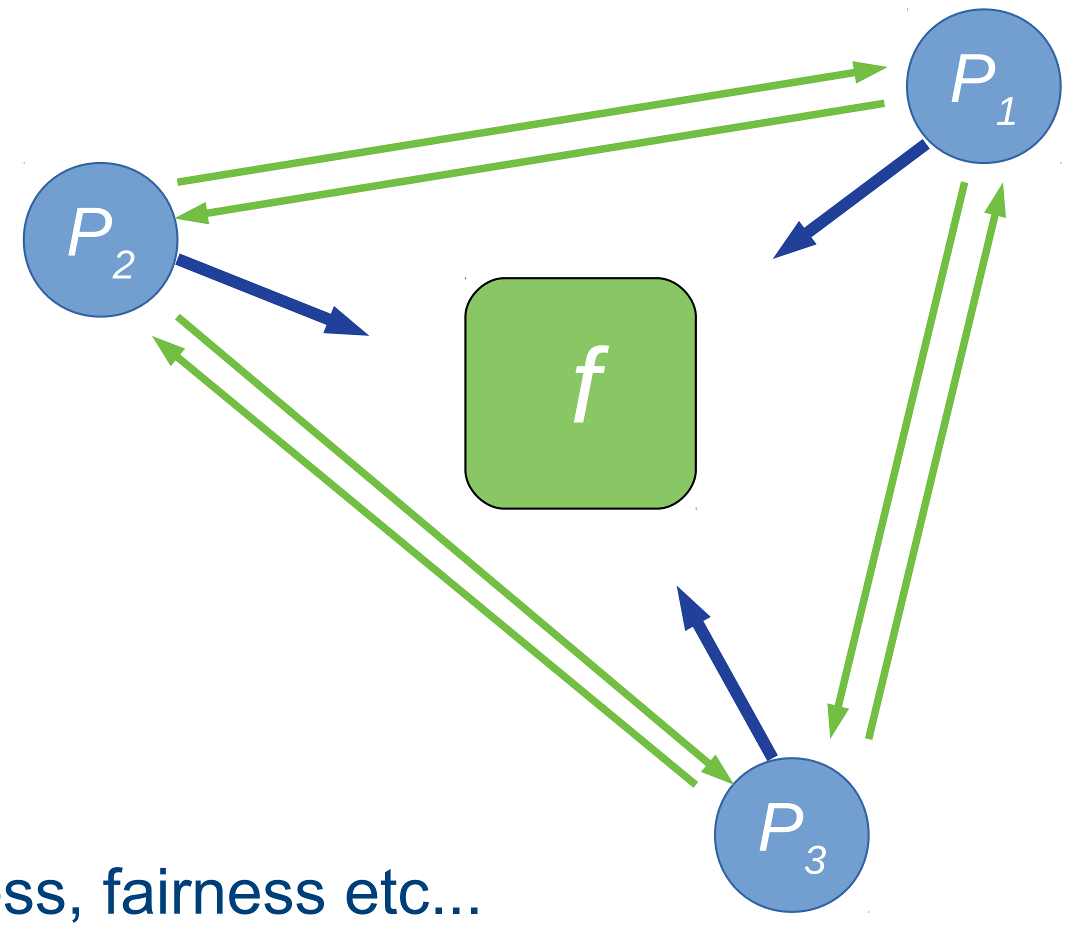
[2]University of Bristol
Department of Computer Science

[3]Aarhus University
Department of Computer Science

Outline

- Multiparty computation
- SPDZ
- Achieving active security with zero-knowledge proofs
- The zero-knowledge proof in SPDZ
- Improving zero-knowledge proof: TopGear
- Experimental results

Multiparty computation



- Secure: privacy, correctness, fairness etc...

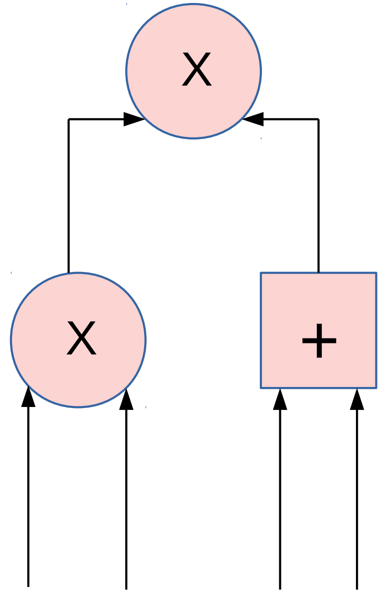
SPDZ preprocessing model

Preprocessing

Public key algorithms

Randomness
(Beaver triples)

Circuit evaluation



Information theoretic primitives

KU LEUVEN

The preprocessing phase

- Multiplications computed using Beaver method
  - Beaver triples: (a,b,c) with a,b randomly selected from R_p and $c = ab$.
  - Triples are authenticated and shared among parties
- This is done by using a distributed (somewhat) homomorphic scheme (BGV)
  - Ring-LWE scheme with plaintext space $R_p = \mathbb{Z}_p[X] / (X^N+1)$

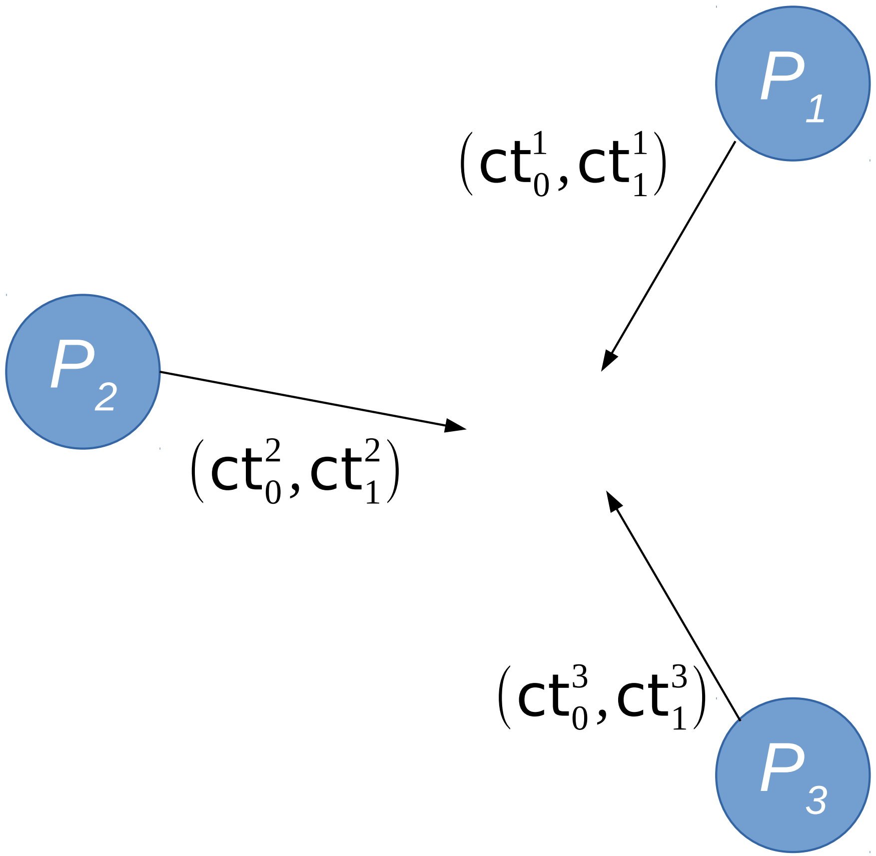
KU LEUVEN

Beaver triples

1. P_i samples a_i, b_i, c_i
2. P_i computes $\texttt{BGV.Enc}(a_i)$, $\texttt{BGV.Enc}(b_i)$, $\texttt{BGV.Enc}(c_i)$
3. P_i proves knowledge of $\texttt{BGV.Enc}(a_i)$, $\texttt{BGV.Enc}(b_i)$, $\texttt{BGV.Enc}(c_i)$ and broadcasts them
4. Parties compute $\mathrm{ct}_c = (\sum \texttt{BGV.Enc}(a_i))(\sum \texttt{BGV.Enc}(b_i)) - (\sum \texttt{BGV.Enc}(c_i))$, with $\mathrm{ct}_c = \texttt{BGV.Enc}(\gamma)$ for some γ.
5. Parties jointly run $\texttt{BGV.Dec}(\mathrm{ct}_c)$ and get $\gamma = \texttt{BGV.Dec}(\mathrm{ct}_c)$
6. P_1 's shared triple is $(a_1, b_1, c_1 + \gamma)$
7. P_i 's shared triple is (a_i, b_i, c_i)
8. At the end of the process each party holds a share of the triple a, b, c

Preprocessing phase

$$\mathtt{BGV.Enc}(x^i; r^i) = (ct_0^i ct_1^i) = (ct_0^i, ct_0^i x^i + r^i)$$



To avoid fault attacks, need to check each BGV ciphertext is well formed

- Noise must be bounded
- Prove that $||x^i||, ||r^i|| < B$

KU LEUVEN

Schnorr-like approach

Prover[*i*]		**Verifier**
$y^i, t^i \leftarrow \mathcal{D}$	$\mathtt{com}_i = \mathtt{ct}_0^i \cdot y^i + t^i$ $\longrightarrow$	
	$\longleftarrow$ $\mathtt{chal}_i = e$	$e \in \{0,1\}$
$\epsilon^i = t^i + e\, r^i$ $\sigma^i = y^i + e\, x^i$	$\mathtt{resp}_i = \sigma^i, \epsilon^i$ $\longrightarrow$	$\mathtt{ct}_0^i \cdot \sigma^i + \epsilon^i = (\mathtt{ct}_0^i \cdot y^i + t^i) + e \cdot \mathtt{ct}_1^i$ $\|\sigma^i\|, \|\epsilon^i\| < 2B'$

One ciphertext per run, no auxiliary ciphertexts

repeat *n* times

Schnorr-like approach

- Issue 1
  - Not zero-knowledge unless Prover masks the secret with large values
  - Blows up parameters
  - Introduces the adversarial language

    $\mathscr{Z}' = \{ (x, r) \text{ s.t. } ||x||, ||r|| \leq 2^{\texttt{ZK\_sec}} B \}$
  - As opposite to the honest language

    $\mathscr{Z} = \{ (x, r) \text{ s.t. } ||x||, ||r|| \leq B \}$
  - $\mathscr{Z}' / \mathscr{Z}$ = slack

Schnorr-like approach

- Issue 2
  - Binary challenge set
  - Dishonest prover has ½ chances to cheat
  - Need to repeat the protocol $\texttt{sec}$ times to achieve $2^{-\texttt{sec}}$ soundness security
  - In general, enlarging the challenge space does not help

    we only prove $\mathscr{Z}' = \{\, (x, r) \text{ s.t. } \kappa \cdot ||x||, \kappa \cdot ||r|| \leq 2^{\texttt{ZK\_sec}} B \,\}$

SPDZ'12 – amortization

Prover[*i*]

$y^i, t^i \leftarrow \mathcal{D}$

$\mathtt{com}_i = \mathtt{ct}_0^i \cdot y^i + t^i$ →

← $\mathtt{chal}_i = e$

$\epsilon^i = t^i + e\, r^i$

$\sigma^i = y^i + e\, x^i$

$\mathtt{resp}_i = \sigma^i, \epsilon^i$ →

Verifier

$$e \leftarrow \begin{pmatrix} \star & \star & 0 & 0 \\ 0 & \star & \star & 0 \\ 0 & 0 & \star & \star \end{pmatrix}$$

$$\mathtt{ct}_0^i \cdot \sigma^i + \epsilon^i = (\mathtt{ct}_0^i\, y^i + t^i) + e\, \mathtt{ct}_1^i$$

$$\|\sigma^i\|, \|\epsilon^i\| < 2B'$$

U = `sec` ciphertexts at once, $V = 2U - 1$ auxiliary ciphertexts

Repeat n times

SPDZ'12 – amortization

- Still binary challenge space
- Slack increases by $2^{\texttt{sec}}$
- Needs $\texttt{sec} + U$ ciphertexts for proving U ciphertexts
- Memory consumption roughly $3U = 3\texttt{sec}$
- Need to keep U small
  - Limits security

Beaver triples

1. P_i samples a_i, b_i, c_i
2. P_i computes $\texttt{BGV.Enc}(a_i)$, $\texttt{BGV.Enc}(b_i)$, $\texttt{BGV.Enc}(c_i)$
3. P_i proves knowledge of $\texttt{BGV.Enc}(a_i)$, $\texttt{BGV.Enc}(b_i)$, $\texttt{BGV.Enc}(c_i)$ and broadcasts them
4. Parties compute $\mathrm{ct}_c = (\sum \texttt{BGV.Enc}(a_i))(\sum \texttt{BGV.Enc}(b_i)) - (\sum \texttt{BGV.Enc}(c_i))$, with $\mathrm{ct}_c = \texttt{BGV.Enc}(\gamma)$ for some γ.
5. Parties jointly run $\texttt{BGV.Dec}(\mathrm{ct}_c)$ and get $\gamma = \texttt{BGV.Dec}(\mathrm{ct}_c)$
6. P_1's shared triple is $(a_1, b_1, c_1 + \gamma)$
7. P_i's shared triple is (a_i, b_i, c_i)
8. At the end of the process each party holds a share of the triple a, b, c

KU LEUVEN

Overdrive'18 – sum of statements

Prover[i]		**Verifier**
$y^i, t^i \leftarrow D$	$\mathtt{com}_i = \mathtt{ct}_0^i \cdot y^i + t^i$ $\longrightarrow$	
	$\longleftarrow$ $\mathtt{chal} = e$	$e \leftarrow \begin{pmatrix} \star & \star & 0 & 0 \\ 0 & \star & \star & 0 \\ 0 & 0 & \star & \star \end{pmatrix}$
$\epsilon^i = t^i + e\, r^i$ $\sigma^i = y^i + e\, x^i$	$\mathtt{resp}_i = \sigma^i, \epsilon^i$ $\longrightarrow$	$\mathtt{ct}_0^i \sum_{i=1}^{n} \sigma^i + \sum_{i=1}^{n} \epsilon^i = \sum_{i=1}^{n} (\mathtt{ct}_0^i \cdot y^i + t^i) + e \sum_{i=1}^{n} \mathtt{ct}_1^i$ $\|\sum \sigma^i\|, \|\sum \epsilon^i\| < 2nB'$

U = `sec` ciphertexts at once, $V = 2U - 1$ auxiliary ciphertexts

Do it only once

Overdrive – sum of statements

- Do not need to repeat the protocol
- Amortization as in CD09
  - High soundness slack
- Memory consumption is still roughly $3U = 3\texttt{sec}$
- Security not optimal
  - High security parameter means more memory usage

TopGear

Prover[i]		**Verifier**
$y^i, t^i \leftarrow D$	$\mathtt{com}_i = \mathtt{ct}_0^i \cdot y^i + t^i$ $\rightarrow$	
	$\leftarrow$ $\mathtt{chal} = e$	$e \leftarrow \{\pm X^i \cup \{0\}\}^{V \times U}$
$\epsilon^i = t^i + e\, r^i$ $\sigma^i = y^i + e\, x^i$	$\mathtt{resp}_i = \sigma^i, \epsilon^i$ $\rightarrow$	$\mathtt{ct}_0^i \sum_{i=1}^{n} \sigma^i + \sum_{i=1}^{n} \epsilon^i = \sum_{i=1}^{n} (\mathtt{ct}_0^i\, y^i + t^i) + e \sum_{i=1}^{n} \mathtt{ct}_1^i$ $\lVert \sum \sigma^i \rVert, \lVert \sum \epsilon^i \rVert < 2nB'$

$U = 2V$ ciphertexts at once, $V = \mathtt{sec} / \log(2N{+}1)$

Do it only once

TopGear

- Larger challenge space
  - Use $\{ \pm X^i \cup \{0\} \}_{i = 1,\ldots,2N}$
    - Extraction as in BBC[+]18
    - Uses crucially the fact that in R_q the element $2/(X^i-X^j)$ has norm less than 1
- Better amortization while maintaining the same level of security
  - $V = \texttt{sec} / \log_2(2N+1)$, $U = 2V$
  - N typically of 15 bits
- More triples

KU LEUVEN

TopGear

- We do not care about slack
  - Distributed decryption uses modulo switching operations which reduces the slack
  - Even better with new amortization
- We can fix extraction
  - Proof lies in $\mathscr{Z}' = \{\ (x, r)\ \text{ s.t. }\ 2\cdot\|x\|, 2\cdot\|r\| \leq 2^{\mathtt{ZK\_sec}} B\ \}$
  - Scheme is homomorphic hence multiply ciphertexts by 2

Implementation

- SCALE-MAMBA v. 1.4
  - https://github.com/KULeuven-COSIC/SCALE-MAMBA
- TopGear
  - https://github.com/KULeuven-COSIC/SCALE-MAMBA/src/FHE/ZKPoK.cpp

Memory consumption (2 parties, `ZK_sec` = `sec` = 40)[1]

t_{Tr} \ t_{ZK}	1	2	4	8
1	25	41	68	98
2	25	38	68	98
4	28	49	75	98
8	32	52	81	98

Overdrive

t_{Tr} \ t_{ZK}	1	2	4	8
1	7	9	15	27
2	8	10	15	26
4	10	12	17	28
8	14	17	21	33

TopGear

[1] Tested on i7-7700K CPUs in a LAN setting.

Memory consumption (2 parties, `ZK_sec` = `sec` = 128)[1]

t_{Tr} \ t_{ZK}	1	2	4	8
1	70	98	-	-
2	72	98	-	-
4	73	98	-	-
8	76	98	-	-

Overdrive

t_{Tr} \ t_{ZK}	1	2	4	8
1	11	19	33	63
2	12	18	33	64
4	14	21	34	64
8	16	24	39	70

TopGear

[1] Tested on i7-7700K CPUs in a LAN setting.

Triples per second (2 parties, `ZK_sec = sec` = 40)[1]

t_{Tr} \ t_{ZK}	1	2	4	8
1	1503	1602	1562	1335
2	1488	2347	2212	1976
4	1272	1876	2150	1865
8	976	1307	1464	1533

Overdrive

t_{Tr} \ t_{ZK}	1	2	4	8
1	2806	2829	2846	2752
2	3809	4851	4709	4540
4	5672	6086	6692	6293
8	4666	5635	6084	5636

TopGear

[1] Tested on i7-7700K CPUs in a LAN setting.

Triples per second (2 parties, `ZK_sec = sec` = 128)[1]

t_{Tr} \ t_{ZK}	1	2	4	8
1	1240	1369	-	-
2	1426	1834	-	-
4	1231	1612	-	-
8	940	1129	-	-

Overdrive

t_{Tr} \ t_{ZK}	1	2	4	8
1	1743	2775	2692	2569
2	4251	4622	4572	4021
4	3943	3712	4955	5000
8	3265	3272	5254	5041

TopGear

[1] Tested on i7-7700K CPUs in a LAN setting.

Questions?

https://eprint.iacr.org/2019/035

KU LEUVEN