Efficient Constant-Round Multiparty Computation

Yehuda Lindell

Bar-Ilan University

Based on joint works with Aner Ben-Efraim, Eran Omri, Benny Pinkas, Nigel Smart, Eduardo Soria-Vasquez and Avishai Yanay

BIU Center for Research in Applied Cryptography and Cyber Security

The Search for the Fastest Protocol

- Ideally – a best/fastest protocol
- In reality – it depends on the requirements and setting
- The main parameters:
  - Computational power (in this talk we'll assume standard machines)
  - **Network speed**: LAN vs WAN
- The requirements:
  - Security level (semi-honest, covert, malicious)
  - Speed: **low latency** or **high throughput**
    - Note: online/offline really only helps for latency (or for settings where throughput demands have high variance)

Two Main Paradigms for Secure Computation

The garbled-circuit paradigm

- Constant-round
- High bandwidth

- **Conclusion**:
  - Suitable for low latency goal
  - Performs well even in slow networks
  - High bandwidth means low throughput

The secret-sharing paradigm

- Many rounds (depth of circuit)
- Low bandwidth

- **Conclusion**:
  - Suitable for high throughput goal
  - Performs well on fast networks only
  - Multiple rounds means bad performance for deep circuits

Some Sample Numbers – SHA256

- Circuit parameter: the SHA256 circuit has almost 100,000 AND gates and has depth 4000
- Garbled circuits:
  - The best garbled circuit has 256 bits per AND gate
  - The size of the garbled circuit is 25Mb
  - On a 10Gbps connection, cannot send more than **400 circuits per second**
- Secret sharing:
  - On a 30ms latency network, minimum computation latency **120 seconds**
  - On a 1ms latency network, minimum computation latency 4 seconds
  - On a 0.1ms latency network, minimum computation latency 0.4 seconds

Concrete Efficiency – The Last Decade

- Two-party computation (semi-honest)
  - Fairplay (2004): 4383 gates, 7.09 seconds on a LAN
  - Long series of works: Yao, GMW, OT extensions
  - Latest (2014): 22,000 gates (6800 AND), 16ms on a LAN
  - Improvement factor of 2000; Moore's law gives 32
- Two-party computation (malicious)
  - Long series of works: cut-and-choose Yao, LEGO-type, SPDZ, TinyOT,…
- Work on semi-honest has been significant in malicious setting
  - Faster and smaller garbled circuits, OT extensions, circuit optimizations…

Concrete Efficiency – The Last Decade

- Multiparty computation (semi-honest)
  - FairplayMP (2008): 1024 gates, 10 sec for 5 parties, 55 sec for 10 parties
    - But only honest majority
  - GMW implementation (CHKMR 2012): 5500 gates, 7 sec for 5 parties, 10 sec for 10 parties (but actually much faster)
- Multiparty computation (malicious)
  - SPDZ, multiparty TinyOT
- Almost no work in the semi-honest setting: **since FairplayMP nothing constant-round** (for low latency goal even in slow networks)

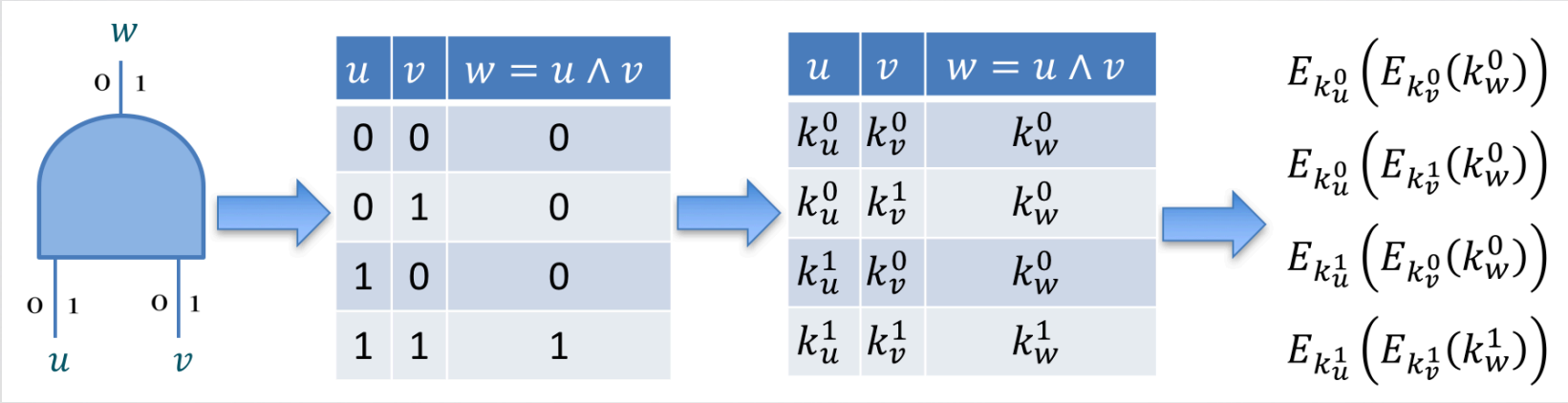
Multiparty Computation

- Secret sharing approach:
  - Information-theoretic protocols
  - GMW
  - Suitable for high throughput

- Garbled-circuit approach:
  - The BMR protocol (Beaver-Micali-Rogaway), constant round
  - Potential for low latency

The BMR Garbled Circuit

- Background – Yao's garbled circuits



u	v	$w = u \wedge v$
0	0	0
0	1	0
1	0	0
1	1	1

u	v	$w = u \wedge v$
k_u^0	k_v^0	k_w^0
k_u^0	k_v^1	k_w^0
k_u^1	k_v^0	k_w^0
k_u^1	k_v^1	k_w^1

$$E_{k_u^0}\left(E_{k_v^0}(k_w^0)\right)$$
$$E_{k_u^0}\left(E_{k_v^1}(k_w^0)\right)$$
$$E_{k_u^1}\left(E_{k_v^0}(k_w^0)\right)$$
$$E_{k_u^1}\left(E_{k_v^1}(k_w^1)\right)$$

- Relies inherently on the fact that one party garbles and the other evaluates
- Cannot work this way in the multiparty setting (collusions!)

The BMR Garbled Circuit

- The idea – each party contributes a secret to mask each value
  - Let u, v be input wires and let w be output wire; let n be number of parties
  - Each party P_i chooses random keys $k_{x,i}^0$ and $k_{x,i}^1$ for each wire $x \in \{u, v, w\}$
  - For every $a, b \in \{0,1\}$ and every $i \in \{1, \ldots, n\}$, double-encrypt $k_{w,i}^{g(a,b)}$ under the keys $k_{u,1}^a, k_{u,2}^a, \ldots, k_{u,n}^a$ and $k_{v,1}^b, k_{v,2}^b, \ldots, k_{v,n}^b$

- Using a PRG: $c_{a,b} = \bigoplus_{i=1}^{n} \left(G(k_{u,i}^a) \oplus G(k_{v,i}^b) \right) \oplus \left(k_{w,1}^{g(a,b)} || \cdots || k_{w,n}^{g(a,b)} \right)$

- Using a PRF: $\forall j \;\; c_{a,b}^j = \bigoplus_{i=1}^{n} \left(F_{k_{u,i}^a}(g||j) \oplus F_{k_{v,i}^b}(g||j) \right) \oplus k_{w,j}^{g(a,b)}$

Point and Permute

- For every wire u, parties generate a secret random $\lambda_u \in \{0,1\}$
- The value $\lambda_u \oplus \alpha$ is revealed, where α is the real value on the wire
- On input wires, if u is associated with P_i's input, then it receives λ_u
- On output wires, λ_u is made public

- The actual ciphertext equation:

$$\forall j \in [n]: \; c_{a,b}^{j} = \bigoplus_{i=1}^{n} \left(F_{k_{u,i}^{a}}(g||j) \oplus F_{k_{v,i}^{b}}(g||j) \right) \oplus k_{w,j}^{\boldsymbol{g(a\oplus\lambda_u, b\oplus\lambda_v)\oplus\lambda_w}}$$

The Original BMR Protocol

- **Primary observation**: given keys on all wires, the circuit needed to construct the BMR circuit is of constant depth

- Use any existing protocol with rounds=O(depth) to securely compute the BMR circuit
  - Semi-honest: use GMW; each party inputs result of PRF computations
  - Malicious: need to work harder; BMR only did honest majority
    - Use general compiler from semi-honest to malicious
    - Need to be constant round (so coin-tossing of Pass)

The Aim

- Optimize BMR in the semi-honest setting
  - Joint work with Aner Ben-Efraim and Eran Omri
- Construct a BMR protocol for the malicious setting
  - Using SPDZ – joint work with Benny Pinkas, Nigel Smart and Avishay Yanay (CRYPTO 2015)
  - Using SHE directly – joint work with Nigel Smart and Eduardo Soria-Vazquez

FairplayMP

- Used BGW to compute the equation for the garbled gate
  - Map the concatenation of all keys to a single field element
  - Natural over an arithmetic circuit

- Drawbacks of approach:
  - Only for an honest majority (uses BGW)
  - Very large field computations

Optimizing Semi-Honest BMR

- Main contributions:
  - Adapt free-XOR (when using arithmetic circuit, requires a characteristic-2 field)
  - Construct a protocol based on OT (no honest majority)
  - Construct faster BGW-based protocols
    - FairplayMP worked in a prime field; coin flipping of λ_u values is complex
  - Implement and compare to GMW

Computing Garbled Gates

- We translate the equation into an arithmetic circuit
- The equation for gate function $g(a,b)$:

$$c_{a,b}^{j} = \begin{cases} \oplus_{i=1}^{n} \left(F_{k_{u,i}^{a}}(g||j) \oplus F_{k_{v,i}^{b}}(g||j) \right) \oplus k_{w,j}^{0} & \text{if } g(a,b) = \lambda_w \\ \oplus_{i=1}^{n} \left(F_{k_{u,i}^{a}}(g||j) \oplus F_{k_{v,i}^{b}}(g||j) \right) \oplus k_{w,j}^{1} & \text{if } g(a,b) \neq \lambda_w \end{cases}$$

Computing Garbled Gates

- We translate the equation into an arithmetic circuit
- The equation for an **AND** gate:

$$c^{j}_{a,b} = \begin{cases} \oplus_{i=1}^{n} \left(F_{k^{a}_{u,i}}(g||j) \oplus F_{k^{b}_{v,i}}(g||j) \right) \oplus k^{0}_{w,j} & \text{if } (a \oplus \lambda_u) \cdot (b \oplus \lambda_v) = \lambda_w \\ \oplus_{i=1}^{n} \left(F_{k^{a}_{u,i}}(g||j) \oplus F_{k^{b}_{v,i}}(g||j) \right) \oplus k^{1}_{w,j} & \text{if } (a \oplus \lambda_u) \cdot (b \oplus \lambda_v) \neq \lambda_w \end{cases}$$

Computing Garbled Gates

- We translate the equation into an arithmetic circuit
- The equation for an **AND** gate:

$$c_{a,b}^{j} = \bigoplus_{i=1}^{n} \left(F_{k_{u,i}^{a}}(g||j) \oplus F_{k_{v,i}^{b}}(g||j) \right) \oplus \begin{cases} k_{w,j}^{0} & \text{if } (a \oplus \lambda_u) \cdot (b \oplus \lambda_v) = \lambda_w \\ k_{w,j}^{1} & \text{if } (a \oplus \lambda_u) \cdot (b \oplus \lambda_v) \neq \lambda_w \end{cases}$$

Arithmetizing the Expression

- The equation for AND:

$$c_{a,b}^{j} = \oplus_{i=1}^{n} \left(F_{k_{u,i}^{a}}(g||j) \oplus F_{k_{v,i}^{b}}(g||j) \right)$$
$$\oplus \left(k_{w,j}^{\lambda_w} \cdot \left(1 - (a \oplus \lambda_u) \cdot (b \oplus \lambda_v)\right) \right)$$
$$\oplus \left(k_{w,j}^{1-\lambda_w} \cdot (a \oplus \lambda_u) \cdot (b \oplus \lambda_v) \right)$$

Free XOR

- For every $i \in [n]$, party P_i chooses a random R_i
- For every wire u, P_i chooses a random $k_{u,i}^0$ and sets $k_{u,i}^1 = k_{u,i}^0 \oplus R_i$
- A side benefit – a much simpler BMR equation!
  - $c_{a,b}^j = \oplus_{i=1}^n \left(F_{k_{u,i}^a}(g||j) \oplus F_{k_{v,i}^b}(g||j) \right)$

    $\oplus\, k_{w,j}^0 \oplus \left(R_j \cdot \left((a \oplus \lambda_u) \cdot (b \oplus \lambda_v) \oplus \lambda_w \right) \right)$

  - This needs 2 instead of 4 multiplications for AND (as well as free for XOR)

A BGW-Based Protocol (the idea)

- $c_{a,b}^{j} = \oplus_{i=1}^{n}\left(F_{k_{u,i}^{a}}(g||j) \oplus F_{k_{v,i}^{b}}(g||j)\right) \oplus k_{w,j}^{0} \oplus \left(R_j \cdot \left((a \oplus \lambda_u) \cdot (b \oplus \lambda_v) \oplus \lambda_w\right)\right)$
- The parties all hold shares of each λ $(\lambda_u^1 \oplus \cdots \oplus \lambda_u^n = \lambda_u)$
- Each party P_i inputs
  - $F_{k_{u,i}^{a}}(g||j) \oplus F_{k_{v,i}^{b}}(g||j)$ for all j $\quad (P_j \text{ inputs } F_{k_{u,j}^{a}}(g||j) \oplus F_{k_{v,j}^{b}}(g||j) \oplus k_{w,j}^{0})$
  - R_i
  - $a \oplus \lambda_u^i$
  - $b \oplus \lambda_v^i$
  - λ_w^i
- Use BGW to compute the result (2 multiplications, 4 additions)

We work in a field large enough for k only (in contrast to FairplayMP)

BGW-Based Protocols

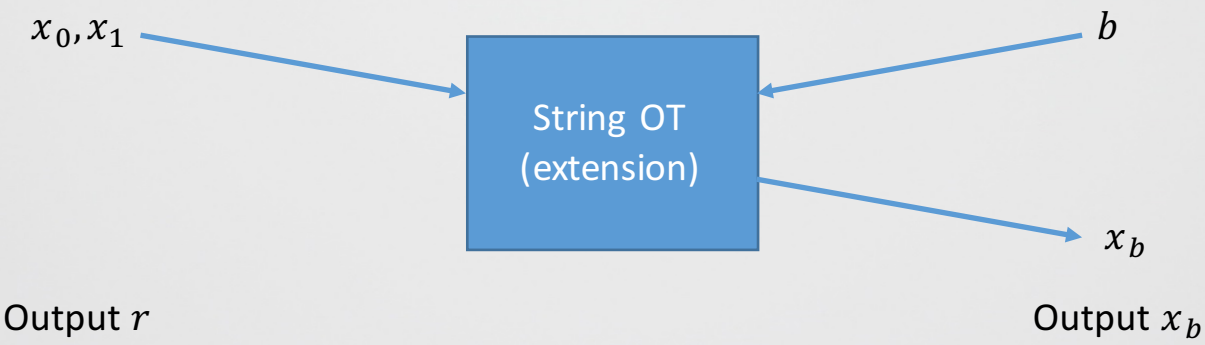
- We have multiple optimizations
  - Fast field multiplication: using PCLMULQDQ and utilizing "small" values
  - Reducing number of rounds: fewer degree reductions
    - The result of $R_j \cdot \left((a \oplus \lambda_u) \cdot (b \oplus \lambda_v) \oplus \lambda_w\right)$ is only added to other values, and so no need to do degree reduction on it
  - And more…
- Complexity: **cubic** in the number of parties
  - Each gate needs n multiplications, but multiplication is quadratic in BGW-semi-honest (computing Shamir shares is $O(n^2)$)

Honest Minority – OT-Based Protocol

- Main observation: we only need to multiply **bits** and a **string by a bit**
- Two-party string-bit multiplication with OT: compute $x \cdot b$

$\boldsymbol{P_1}\ (x)$: Choose random r
Set $x_0 = r;\ x_1 = x \oplus r$

$\boldsymbol{P_2}\ (b)$



Output r

Output x_b

OT-Based Protocol

- Step 1: Compute pairwise XOR shares of $\lambda_u \cdot \lambda_v$
  - This is just the XOR of products $\lambda_u^i \cdot \lambda_v^i$ and so can use bit-OT multiplication
- Step 2: Compute XOR shares of $(a \oplus \lambda_u) \cdot (b \oplus \lambda_v) \oplus \lambda_w$ for each $a, b \in \{0,1\}$ (local computation only)
- Step 3: Compute XOR shares of $R_j \cdot (a \oplus \lambda_u) \cdot (b \oplus \lambda_v) \oplus \lambda_w$
  - This uses a 4 string-OT multiplications between each pair
- Step 4: XOR the result with the PRF values and broadcast

Evaluation

- CREATE (part of DETER):
  - Intel Xeon 2.20GHz, 6 core,
  - Network with 0.1ms ping time (≈ 0.05ms latency)
- Amazon Virginia-Virginia
  - c4.8xlarge instances
  - Network with 1ms ping time (≈ 0.5ms latency)
- Amazon Virginia-Ireland
  - c4.8xlarge instances
  - Network with 75ms ping time (≈37.5ms latency)

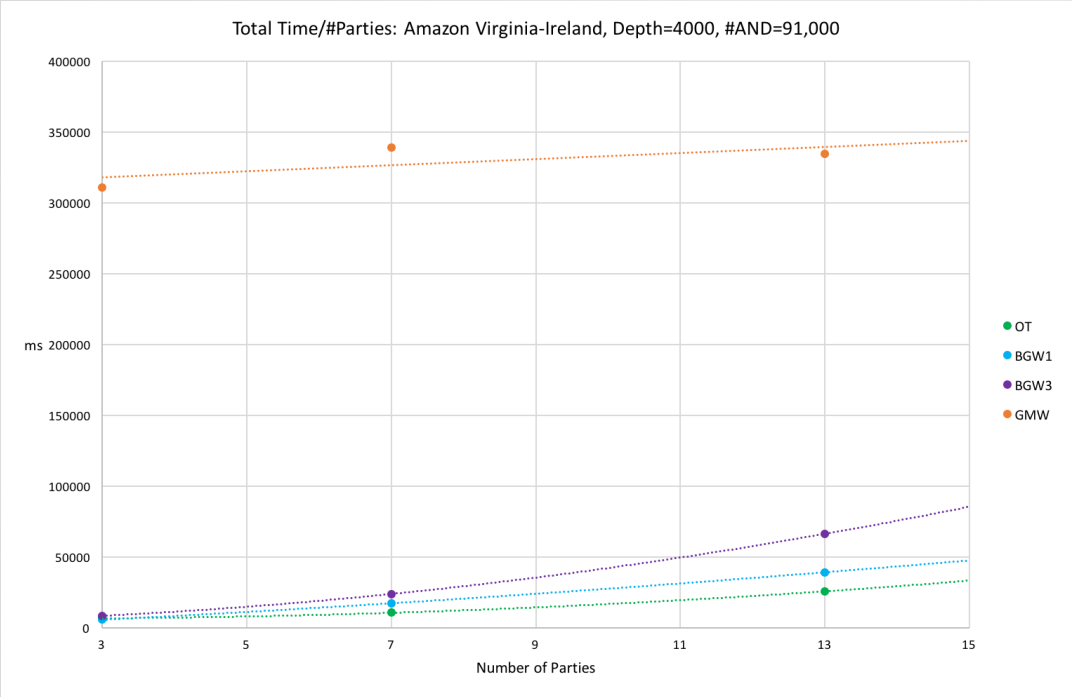
Evaluation

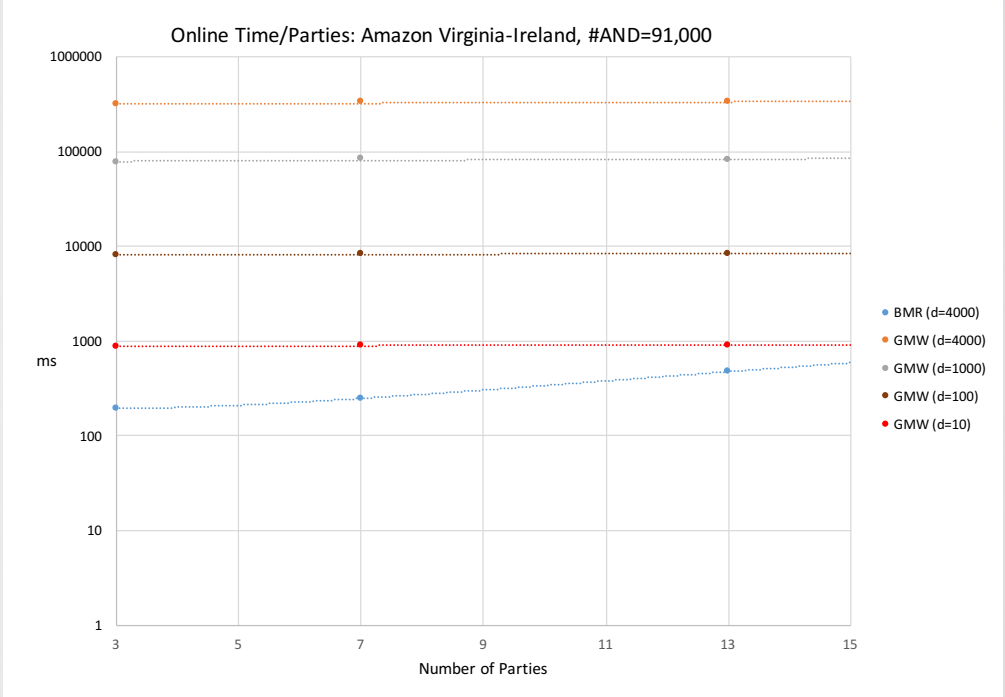
- Compare to GMW in [CHKMR12] on same platforms
  - Uses optimized OT extensions
  - GMW online and offline: OT on random inputs, in online single-bit sent only per AND gate
  - BMR online and offline: build circuit offline, send input and compute online
- Run with:
  - AES circuit: 6800 AND gates, depth = 40
  - SHA256 circuit: 90,825 AND gates, depth = 4000
  - SHA256* synthetic: 90,825 AND gates, depth=10, 100, 1000

Hypotheses

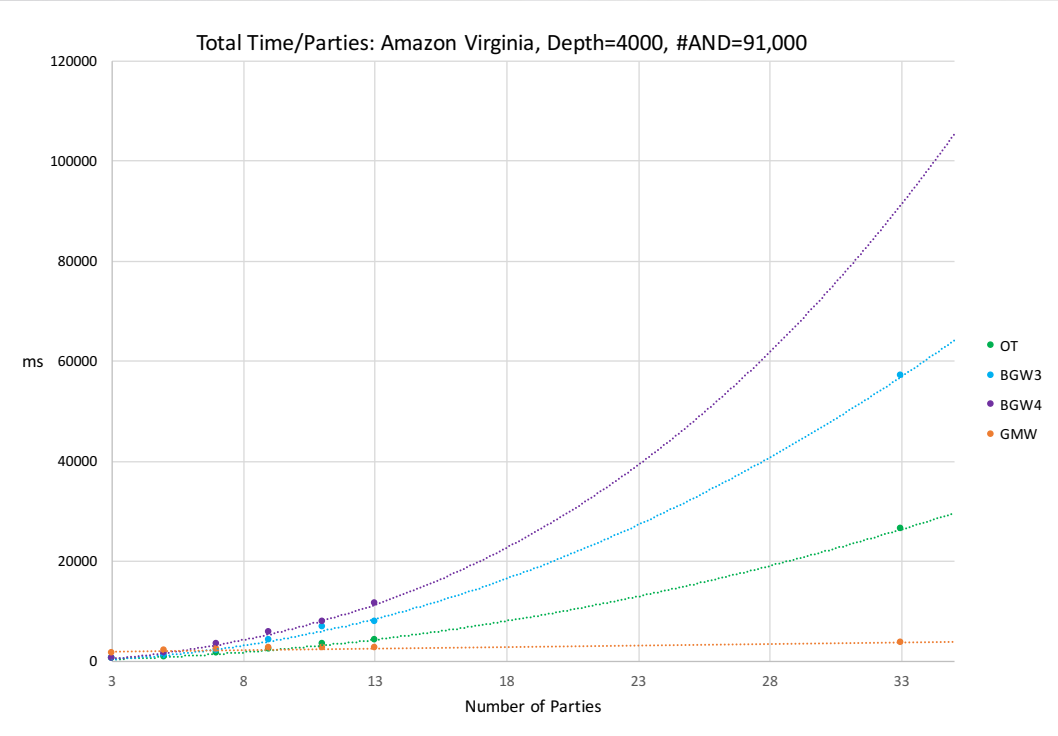
- GMW will win on very shallow circuits in all networks
- BMR will win on deep circuits in all networks
- BMR will win on not shallow circuits in slow networks
- BMR-online will beat GMW-online except for very shallow circuits
- BGW-BMR will beat BGW-OT (but requires honest majority)
- Questions:
  - What is the effect of the number of parties?
  - At what circuit-depth and network speed does BMR/GMW win?

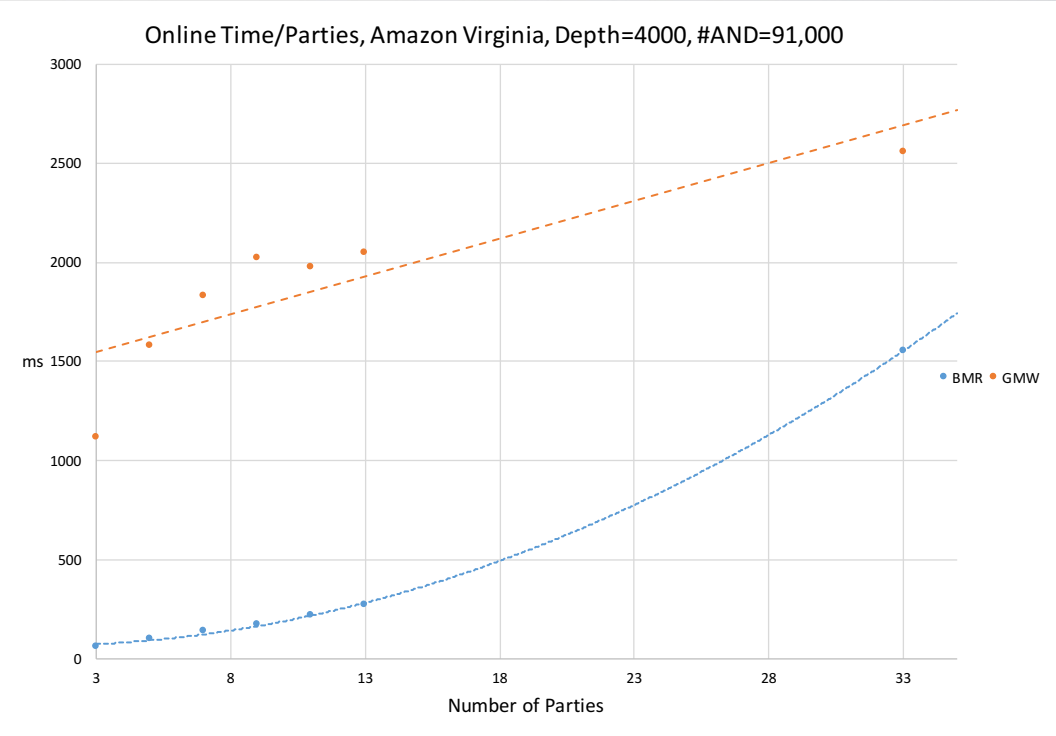
Amazon Virginia-Ireland – WAN (37.5ms latency)



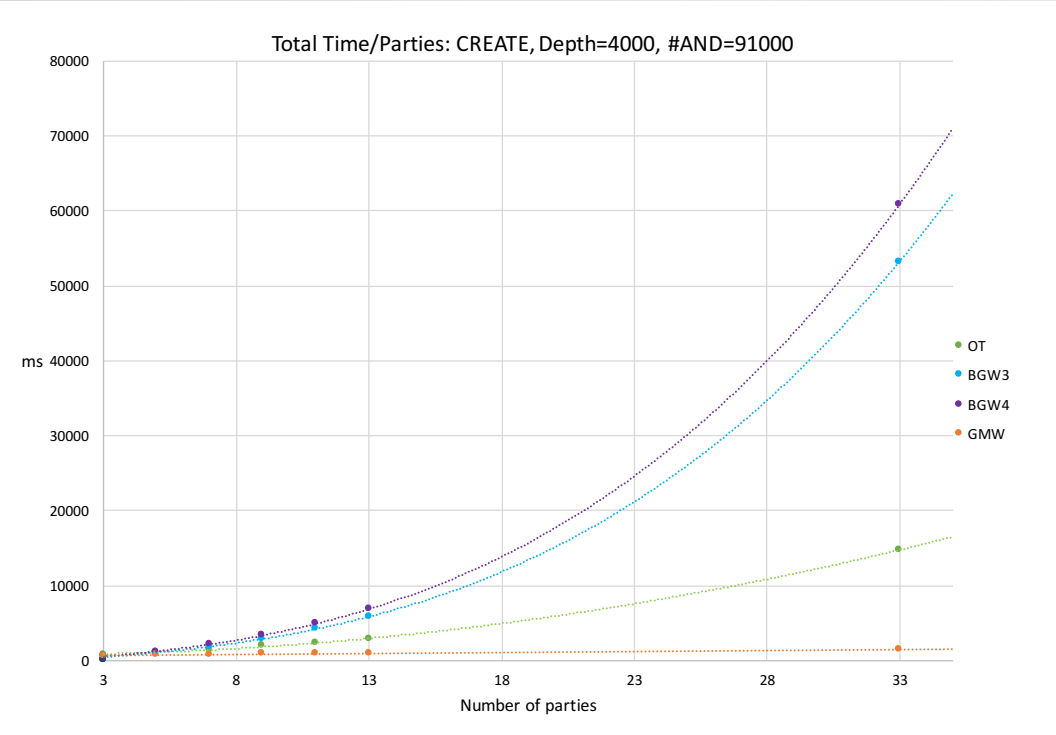


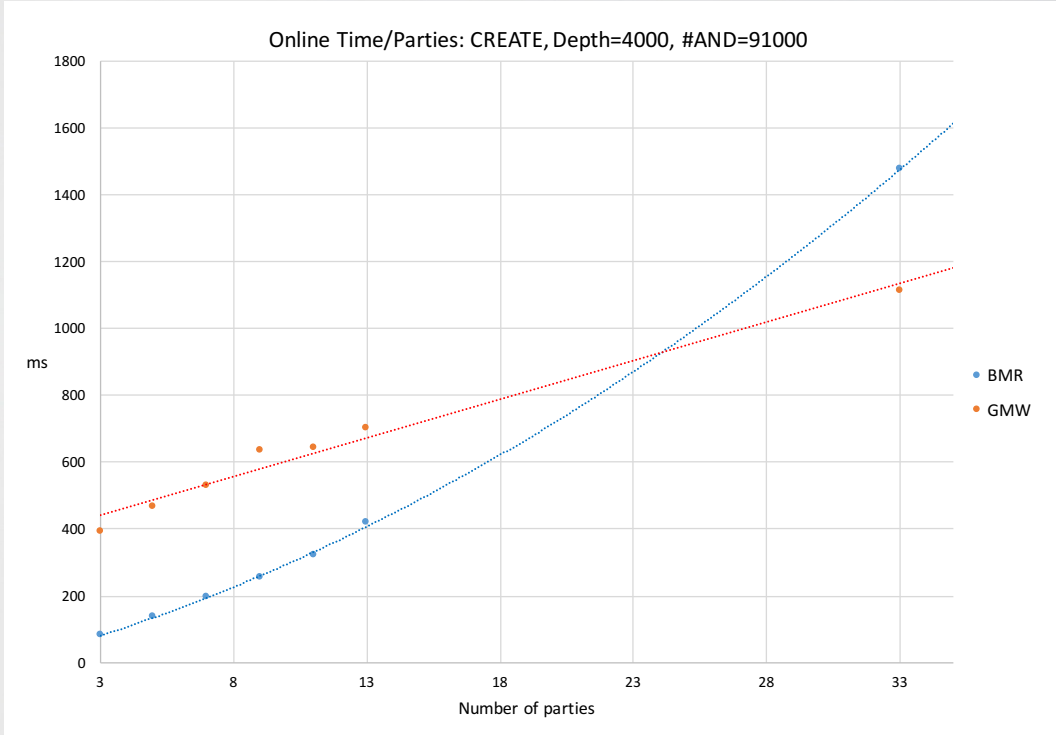
Amazon Virginia – LAN (0.5ms latency)





CREATE – Fast LAN (0.05ms latency)





CREATE – Fast LAN (0.05ms latency)

The SHA256 Circuit – 90,825 AND gates:

		3	5	7	9	11	13	33
OT	Off	813 ± 127	1160 ± 135	1464 ± 106	1963 ± 95	2389 ± 116	2819 ± 122	14928 ± 817
	On	85 ± 15	138 ± 14	204 ± 22	260 ± 28	324 ± 22	419 ± 23	1506 ± 12
BGW3	Off	517 ± 85	1064 ± 154	1864 ± 169	2917 ± 168	4234 ± 192	5825 ± 201	53257 ± 541
	On	81 ± 11	137 ± 15	193 ± 13	252 ± 24	321 ± 39	416 ± 36	1445 ± 130
BGW2	Off		930 ± 118	1799 ± 129	2528 ± 139	3946 ± 179	5690 ± 259	51098 ± 902
	On		135 ± 14	194 ± 13	253 ± 16	315 ± 32	412 ± 47	1485 ± 225
BGW4	Off	582 ± 70	1219 ± 126	2200 ± 193	3383 ± 164	4920 ± 158	6868 ± 170	60858 ± 657
	On	78 ± 11	138 ± 17	196 ± 18	251 ± 18	317 ± 30	419 ± 38	1471 ± 190
GMW	Off	637 ± 67	719 ± 165	789 ± 143	906 ± 261	964 ± 236	953 ± 159	1463 ± 120
(d=4000)	On	391 ± 37	466 ± 140	531 ± 137	636 ± 241	644 ± 196	700 ± 134	1113 ± 81
GMW	Off	674 ± 42	732 ± 170	715 ± 131	873 ± 255	889 ± 212	895 ± 171	1372 ± 158
(d=1000)	On	141 ± 34	187 ± 134	213 ± 138	301 ± 230	314 ± 212	292 ± 169	387 ± 79
GMW	Off	610 ± 42	648 ± 129	755 ± 156	836 ± 242	876 ± 205	870 ± 138	1346 ± 147
(d=100)	On	88 ± 70	105 ± 105	91 ± 88	167 ± 196	176 ± 191	139 ± 134	143 ± 54
GMW	Off	585 ± 76	644 ± 148	716 ± 162	802 ± 223	862 ± 201	857 ± 130	1364 ± 170
(d=10)	On	68 ± 97	70 ± 92	105 ± 246	156 ± 208	124 ± 168	127 ± 150	135 ± 86

BMR wins for few parties only!

Amazon Virginia – LAN (0.5ms latency)

The AES Circuit – 6800 AND gates:

		3	5	7	9	11	13
OT	Off	53 ± 22	91 ± 152	324 ± 1344	429 ± 417	701 ± 1284	1629 ± 3027
	On	6 ± 10	17 ± 17	28 ± 27	43 ± 96	37 ± 24	59 ± 162
BGW3	Off	29 ± 11	103 ± 165	249 ± 364	394 ± 311	838 ± 1305	1008 ± 584
	On	13 ± 15	23 ± 35	28 ± 24	38 ± 20	58 ± 171	59 ± 138
BGW2	Off		88 ± 140	270 ± 322	412 ± 317	670 ± 290	782 ± 339
	On		23 ± 15	33 ± 68	44 ± 78	38 ± 100	46 ± 20
BGW4	Off	47 ± 85	148 ± 243	361 ± 394	682 ± 514	1078 ± 287	1815 ± 2455
	On	8 ± 12	22 ± 16	35 ± 32	36 ± 23	66 ± 206	46 ± 22
GMW	Off	127 ± 47	126 ± 48	125 ± 47	164 ± 186	111 ± 62	116 ± 85
	On	27 ± 11	35 ± 15	43 ± 55	62 ± 142	68 ± 160	119 ± 211

BGW-BMR beats OT-BMR for few parties only;
GMW wins in total time, loses in online time (small circuit)

Amazon Virginia-Ireland – WAN (37.5ms latency)

The AES Circuit – 6800 AND gates:

		3	7	13
OT	Off	698 ± 930	1093 ± 1249	9699 ± 6119
	On	138 ± 88	107 ± 87	362 ± 515
BGW3	Off	329 ± 688	2314 ± 1218	9774 ± 8181
	On	143 ± 81	142 ± 76	329 ± 533
BGW2	Off		2212 ± 1440	8745 ± 6832
	On		148 ± 92	264 ± 409
BGW4	Off	498 ± 737	3149 ± 2065	13298 ± 10576
	On	139 ± 78	159 ± 70	308 ± 473
GMW	Off	231 ± 143	277 ± 1067	382 ± 290
	On	3337 ± 166	3232 ± 9	3341 ± 213

The SHA256 Circuit – 90,825 AND gates:

		3	7	13
OT	Off	6426 ± 1651	10291 ± 4968	25215 ± 4784
	On	172 ± 76	226 ± 62	456 ± 357
BGW3	Off	5404 ± 11751	17011 ± 23574	38584 ± 35997
	On	182 ± 77	237 ± 91	520 ± 659
BGW2	Off		14781 ± 12134	37585 ± 17255
	On		283 ± 86	459 ± 325
BGW4	Off	8124 ± 8000	23521 ± 20794	65736 ± 45895
	On	226 ± 78	282 ± 86	454 ± 281
GMW (d=4000)	Off	850 ± 900	5002 ± 10643	5042 ± 9212
	On	309741 ± 32130	333996 ± 92024	329220 ± 31340
GMW (d=1000)	Off	701 ± 556	3581 ± 4976	7932 ± 16242
	On	77147 ± 4031	83168 ± 19932	82111 ± 5584
GMW (d=100)	Off	735 ± 509	2610 ± 8173	4969 ± 9222
	On	8038 ± 518	8327 ± 80	8341 ± 271
GMW (d=10)	Off	598 ± 362	1180 ± 521	5360 ± 12829
	On	880 ± 75	906 ± 25	904 ± 84

At depth 100, GMW wins in total time even in a WAN, but is an order of magnitude slower in online time

Hypotheses

- GMW will win on very shallow circuits in all networks √
- BMR will win or if deep is 4000, then not true in very fast networks X
- BMR will win (if 100 is not shallow, then true only for few parties (total time) X
- BMR-online will beat GI only for few parties OR deep circuits (in slow network) cuits X
- BGW-BMR will beat BGW-OT (only for few parties onest majority) X
- Questions:
  - What is the effect of the number of parties? marginal in GMW; significant in BMR
  - At what circuit-depth and network speed does BMR/GMW win?

it depends, but GMW far better than expected

Constant-Round for Malicious Adversaries

- The ***only*** multiparty protocol ever implemented for malicious adversaries is SPDZ
  - In a slow network with a deep circuit, this cannot perform well
  - Multiparty TinyOT is also *concretely* efficient, but has many rounds
- Can we use the BMR paradigm in this setting as well?
- A major obstacle: forcing the parties to input the correct PRF values is inherently inefficient (expensive zero knowledge)

SPDZ-BMR [L-Pinkas-Smart-Yanay CRYPTO15]

- Main idea: Use SPDZ to compute the BMR garbled circuit
- Major obstacle – proving correctness of PRF values
- Solution:
  - Don't force the parties to input correct PRF values
  - We prove that inputting incorrect PRF values can only result in **abort**
  - The only problem can be if it changes from one valid value to another
- Obstacle 2 – need to ensure that λ_u values are pseudorandom; coin tossing expensive
- Solution: SPDZ provides coin tossing almost for free

SPDZ-BMR [L-Pinkas-Smart-Yanay CRYPTO15]

- Obstacle 3 – need to force consistency of λ_u^i values when wire u is input to multiple gates
- Solution:
  - Construct a single arithmetic circuit for computing all gates at once
  - Depth of circuit is constant
  - The main goal: **reduce the number of multiplications in the BMR-circuit**

SPDZ-BMR

- The gate computation works as follows:
  - Compute the "indicator variables"

$$[x_a] = \left(f_g([\lambda_a], [\lambda_b]) \stackrel{?}{\neq} [\lambda_c] \right) = (f_g([\lambda_a], [\lambda_b]) - [\lambda_c])^2$$

  - Multiply by the output keys:

$$[\mathbf{v}_{c,x_a}] = (1 - [x_a]) \cdot [\mathbf{k}_{c,0}] + [x_a] \cdot [\mathbf{k}_{c,1}]$$

  - Add in the PRF masks and open:

$$[\mathbf{A}_g] = \sum_{i=1}^{n} \left([F^0_{k^i_{a,0}}(g)] + [F^0_{k^i_{b,0}}(g)] \right) + [\mathbf{v}_{c,x_a}]$$

SPDZ-BMR Cost

- Size of circuit computing the BMR garbled circuit
  - **13** multiplications per AND gate, and **7** multiplications per XOR gate
- Cost of computing the circuit using SPDZ
  - For every wire, need to generate n shared random values
    - Since each gate requires essentially generating n ciphertexts
  - To create a shared random value each of n parties needs to encrypt input data (which must be valid)
  - Each of these requires a ZKPOK, with $O(n)$ SHE encryptions
  - Overall number of SHE multiplications per gate: $O(n^3)$
- Very fast online time – only 2 rounds and local computation

SPDZ-SHE [L-Smart-Soria-Vazquez 2016]

- Main idea: Use somewhat homomorphic encryption (SHE) to **directly** compute the BMR garbled circuit
  - Save the intermediary step of generating multiplication triples
- Major goal: reduce the **depth** of the circuit computing the BMR garbled circuit
  - This has significant influence over the efficiency since it affects the size of the SHE parameters
- We achieve a quadratic number of multiplications only (but need an SHE of depth 3)

SPDZ-SHE

- A naïve approach yields a circuit of depth 4:
  - Multiply to get indicator bit – 2 multiplications (need to square)
  - Multiply indicator bit by keys – 1 more multiplication
  - An additional multiplication is needed (as in SPDZ) to ensure correct output
- Our aim: reduce the depth of the circuit run inside SHE
  - We construct equations multiplying key in directly
  - Our equations do not always compute the correct key
  - Our equation always computes the correct key or its additive complement

SPDZ-BMR

- A depth-2 equation for the AND gate:

$$\langle \mathbf{v}_{c,x_A} \rangle = (1 - \langle \lambda_a \rangle) \cdot \left(\langle \lambda_c \rangle \cdot \langle \tilde{\mathbf{k}}_{c,1} \rangle + (1 - \langle \lambda_c \rangle) \cdot \langle \tilde{\mathbf{k}}_{c,0} \rangle \right) + \langle \lambda_a \rangle \cdot \left((\langle \lambda_b \rangle - \langle \lambda_c \rangle) \cdot \langle \tilde{\mathbf{k}}_{c,1} \rangle + (1 - \langle \lambda_b \rangle - \langle \lambda_c \rangle) \cdot \langle \tilde{\mathbf{k}}_{c,0} \rangle \right)$$

- For example, if $\lambda_a = \lambda_b = \lambda_c = 0$ then we get $k_{c,0}$
- For example, if $\lambda_a = \lambda_b = \lambda_c = 1$ then we get $-k_{c,0}$
- This is a problem:
  - A party learns information if it knows that it received the value or its complement

SPDZ-BMR

- The solution
  - No party knows the basic key values
  - The key used to mask is the **square** of these values
- There is additional cost since the basic key values now need to be generated using an SHE “generate random”
  - Thus, there are more multiplications but the depth is lower

Summary

- The BMR paradigm deserves more attention
- Semi-honest optimizations are an important first step
  - Improvements on the circuit
  - Surprising results regarding the BGW vs OT approaches
- We used SPDZ and SHE to compute for malicious
  - What other methods can be used?