Rabbit: Efficient Comparison for Secure Multi-Party Computation

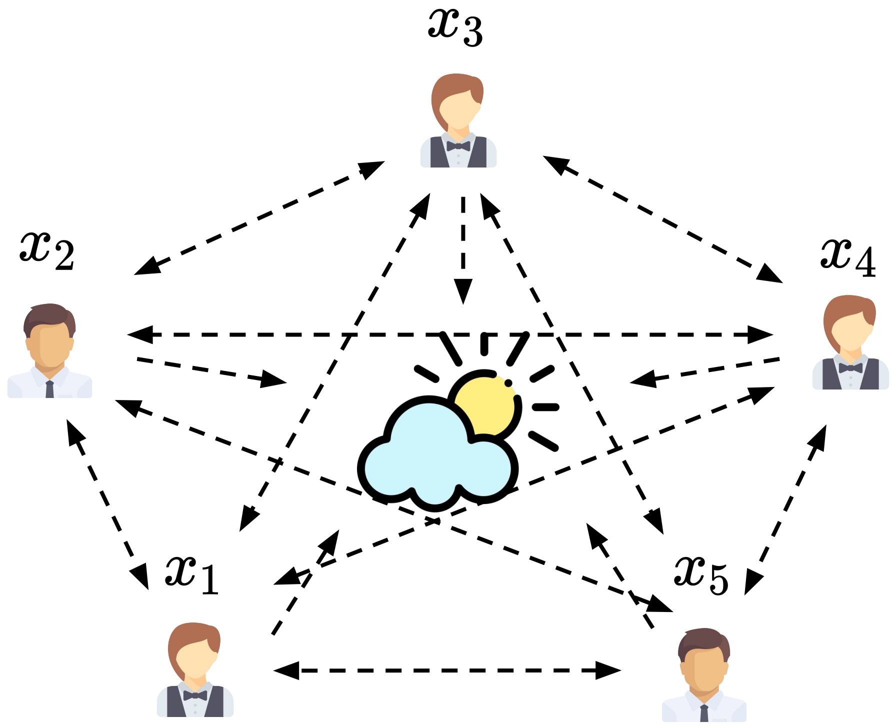
March 1st, 2021
Financial Cryptography and Data Security

Eleftheria Makri[1,5], Dragos Rotaru[2,1], Frederik Vercauteren[1], and Sameer Wagh[3,4]

[1]imec-COSIC, KU Leuven, Belgium
[2]Cape Privacy, Remote
[3]University of California, Berkeley, USA
[4]Princeton University, Princeton, USA
[5]ABRR, Saxion University of Applied Sciences, The Netherlands

emakri@esat.kuleuven.be, dragos@capeprivacy.com,
frederik.vercauteren@kuleuven.be, swagh@berkeley.edu

Multi-Party Computation (Computation with Privacy)



$$F(x_1, x_2, x_3, x_4, x_5)$$

Without revealing

$$x_1, x_2, x_3, x_4, x_5$$

Applications of Multi-Party Computation

Secure comparison

a. k. a Yao's Millionaire problem

$$[x]_M \dashrightarrow (x < R)$$

sharing of $x \in \{0, 1, \cdots, M-1\}$

for a given integer R

$0 \leq R < M$

Threshold

Maximum

$\mathsf{ReLU(x) = max(0, x)}$

Equality

Our Contributions

1

New Comparison Protocols

- New protocols exploiting commutativity of addition
- General n-party protocols
- Arithmetic black box

2

Elimination of Slack

- No gap between the bit-length of the input and the MPC representation
- Enables computation over smaller datatypes

3

Simplicity and Efficiency

- Extremely simple to implement

Standing on the shoulder of giants?

- Prior art [edaBits]: Extended, Doubly-Authenticated Bits

$$\boxed{[x]_M; [x_0]_2, \cdots, [x_{m-1}]_2 \quad \textbf{such that} \quad x = \sum_{i=0}^{m-1} x_i \cdot 2^i}$$

- More effective generation than using m - daBits (improving upon [daBits]) $\longrightarrow$ $[b]_M; [b]_2$

daBit, edaBit ➔ Critical for share conversions

[edaBits] Escudero *et al.* "Improved primitives for MPC over mixed arithmetic-binary circuits." *Annual International Cryptology Conference*, 2020.
[daBits] Rotaru *et al.* “Marbled Circuits: Mixing Arithmetic and Boolean Circuits with Active Security.” *International Conference on Cryptology in India*, 2019.

The Rabbit Collection

$$\begin{tikzcd}
& \Pi_{\text{LTBits}} \arrow[dl] \arrow[dr] & \\
\Pi_{\text{LTS}} & & \Pi_{\text{LTC}} \arrow[d] \\
& & \Pi_{\text{ReLU}}
\end{tikzcd}$$

Comparison of a public constant with a bit decomposed secret
Input : Public constant R and Boolean sharings $[x_0]_2, \cdots, [x_{m-1}]_2$

Comparison of a secret with another secret
Input : Arithmetic sharings $[x]_M$ and $[y]_M$

Comparison of a secret with a constant
Input : Arithmetic sharing $[x]_M$ and public constant R

Specialized comparison for rings with a specific constant
Input : Π_{LTC} when $M = 2^k$ and $R = 2^{k-1}$

The Rabbit Collection

$$\begin{tikzcd}
 & \Pi_{\text{LTBits}} \arrow[dl] \arrow[dr] & \\
\Pi_{\text{LTS}} & & \Pi_{\text{LTC}} \arrow[d] \\
 & & \Pi_{\text{ReLU}}
\end{tikzcd}$$

Comparison of a public constant with a bit decomposed secret
Input : Public constant R and Boolean sharings $[x_0]_2, \cdots, [x_{m-1}]_2$

Comparison of a secret with another secret
Input : Arithmetic sharings $[x]_M$ and $[y]_M$

Comparison of a secret with a constant
Input : Arithmetic sharing $[x]_M$ and public constant R

Specialized comparison for rings with a specific constant
Input : Π_{LTC} when $M = 2^k$ and $R = 2^{k-1}$

The Rabbit Collection

$$\begin{tikzcd}
& \Pi_{\mathsf{LTBits}} \arrow[dl] \arrow[dr] & \\
\Pi_{\mathsf{LTS}} & & \Pi_{\mathsf{LTC}} \arrow[d] \\
& & \Pi_{\mathsf{ReLU}}
\end{tikzcd}$$

Comparison of a public constant with a bit decomposed secret
Input : Public constant R and Boolean sharings $[x_0]_2, \cdots, [x_{m-1}]_2$

Comparison of a secret with another secret
Input : Arithmetic sharings $[x]_M$ and $[y]_M$

Comparison of a secret with a constant
Input : Arithmetic sharing $[x]_M$ and public constant R

Specialized comparison for rings with a specific constant
Input : Π_{LTC} when $M = 2^k$ and $R = 2^{k-1}$

Rabbit Intuition: Observation 1

- Detect when a sum over a particular modulus wraps around and correct for it.
- Given a function: $\mathsf{LT}(\cdot,\cdot) : \mathbb{Z} \times \mathbb{Z} \to \{0,1\} \subseteq \mathbb{Z}$: $\begin{cases} \mathsf{LT}(x,y) = 1 & \text{if } (x < y); \\ \mathsf{LT}(x,y) = 0 & \text{otherwise,} \end{cases}$
- We can compute a modular sum by performing computations over the integers!

$$x + y \bmod M = x + y - M \cdot \mathsf{LT}(x + y \bmod M, x) = x + y - M \cdot \mathsf{LT}(x + y \bmod M, y)$$

Rabbit Intuition: Observation 2

Recall, we are after: $x \overset{?}{<} R$, $R = M - B$

- Exploit the commutativity of addition:

$$\begin{tikzcd}
[x]_M \arrow[r, "+r"] \arrow[d, "+B"] & a = [x + r]_M \arrow[d, "+B"] \\
c = [x + B]_M \arrow[r, "+r"] & b = [x + r + B]_M
\end{tikzcd}$$

- Combine with Observation 1:

$$\begin{aligned} b = [a + B] &= a + B - M \cdot \mathsf{LT}(b, B) \\ &= x + r - M \cdot \mathsf{LT}(a, r) + B - M \cdot \mathsf{LT}(b, B) \end{aligned}$$

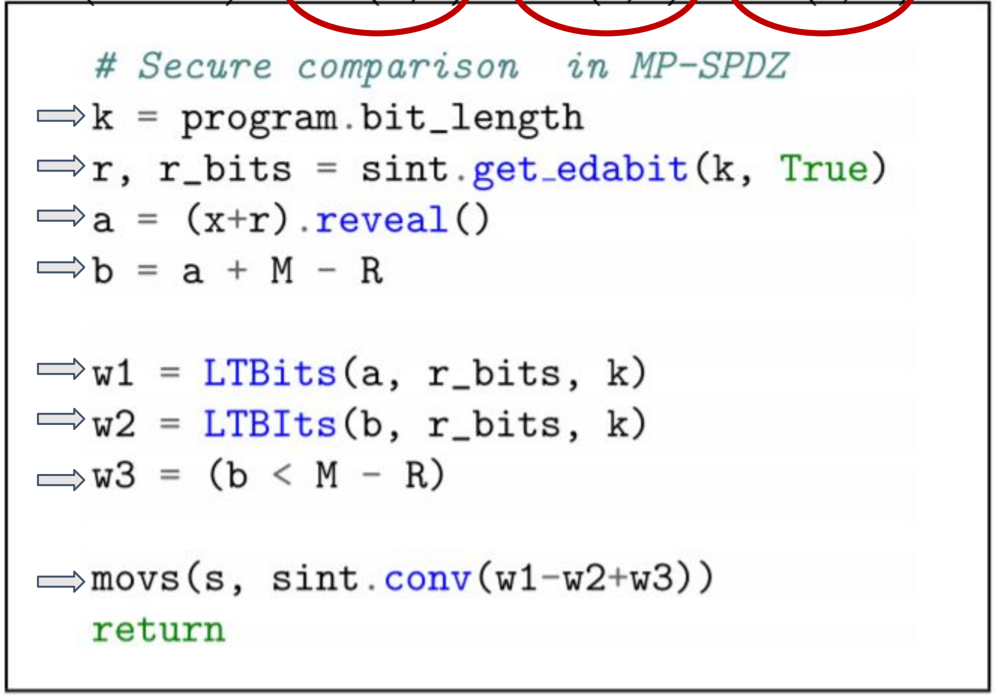
$$\begin{aligned} b = [c + r] &= c + r - M \cdot \mathsf{LT}(b, r) \\ &= x + B - M \cdot \mathsf{LT}(c, B) + r - M \cdot \mathsf{LT}(b, r) \end{aligned}$$

$$\mathsf{LT}(a, r) + \mathsf{LT}(b, B) = \mathsf{LT}(c, B) + \mathsf{LT}(b, r)$$

Rabbit LTC Pseudocode

$$\mathsf{LT}(a, r) + \mathsf{LT}(b, B) = \mathsf{LT}(c, B) + \mathsf{LT}(b, r)$$

$$(x \overset{?}{<} R) = \mathsf{LT}(a, r) - \mathsf{LT}(b, r) + \mathsf{LT}(b, B)$$



```
  # Secure comparison  in MP-SPDZ
k = program.bit_length
r, r_bits = sint.get_edabit(k, True)
a = (x+r).reveal()
b = a + M - R

w1 = LTBits(a, r_bits, k)
w2 = LTBIts(b, r_bits, k)
w3 = (b < M - R)

movs(s, sint.conv(w1-w2+w3))
return
```

The Rabbit Collection

$$\begin{tikzcd}
& \Pi_{\mathsf{LTBits}} \arrow[dl] \arrow[dr] & \\
\Pi_{\mathsf{LTS}} & & \Pi_{\mathsf{LTC}} \arrow[d] \\
& & \Pi_{\mathsf{ReLU}}
\end{tikzcd}$$

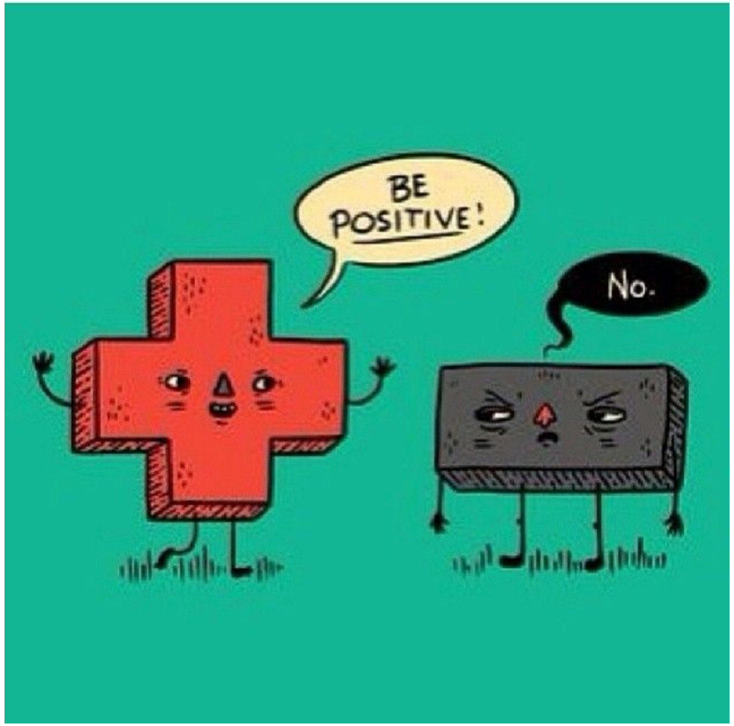
Comparison of a public constant with a bit decomposed secret
Input : Public constant R and Boolean sharings $[x_0]_2, \cdots, [x_{m-1}]_2$

Comparison of a secret with another secret
Input : Arithmetic sharings $[x]_M$ and $[y]_M$

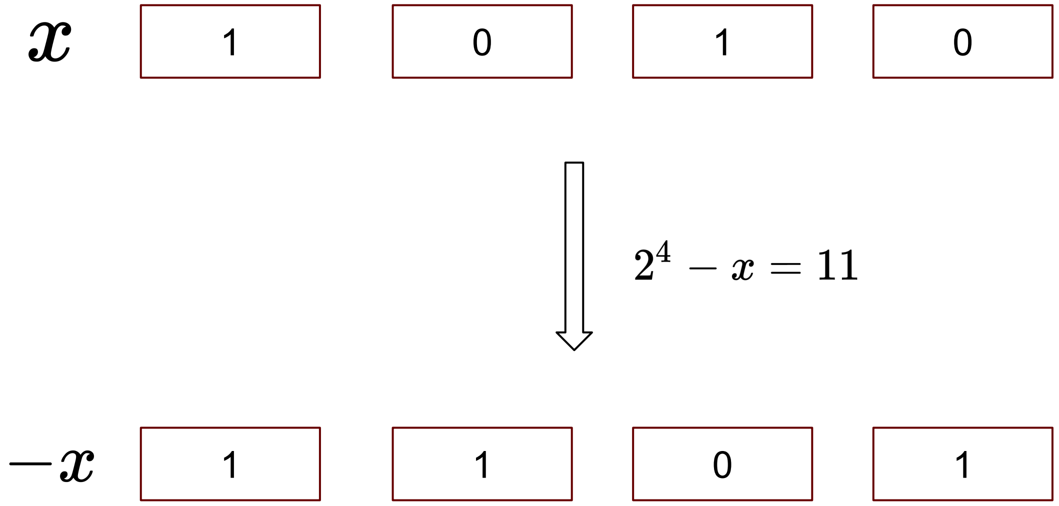
Comparison of a secret with a constant
Input : Arithmetic sharing $[x]_M$ and public constant R

Specialized comparison for rings with a specific constant
Input : Π_{LTC} when $M = 2^k$ and $R = 2^{k-1}$

What is a negative number?



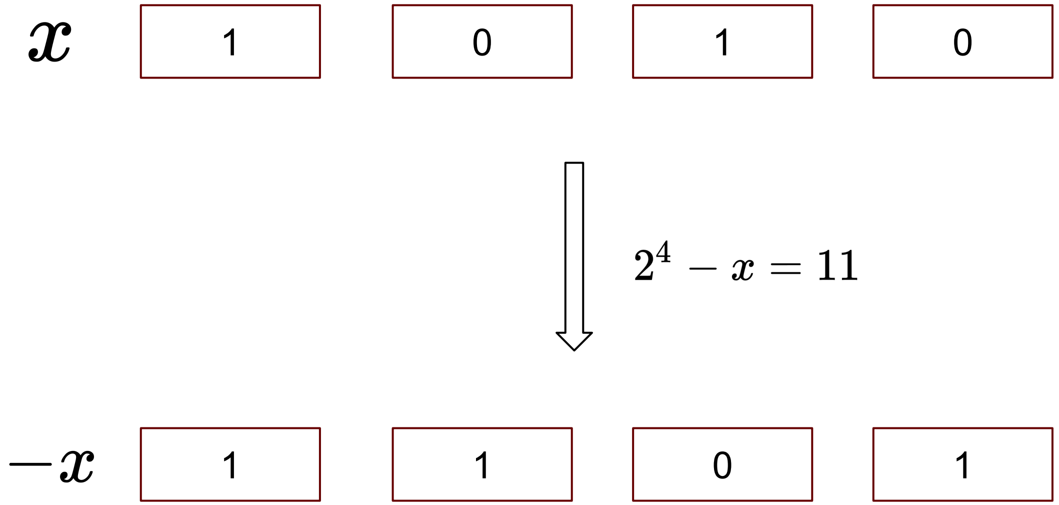
What is a negative number?



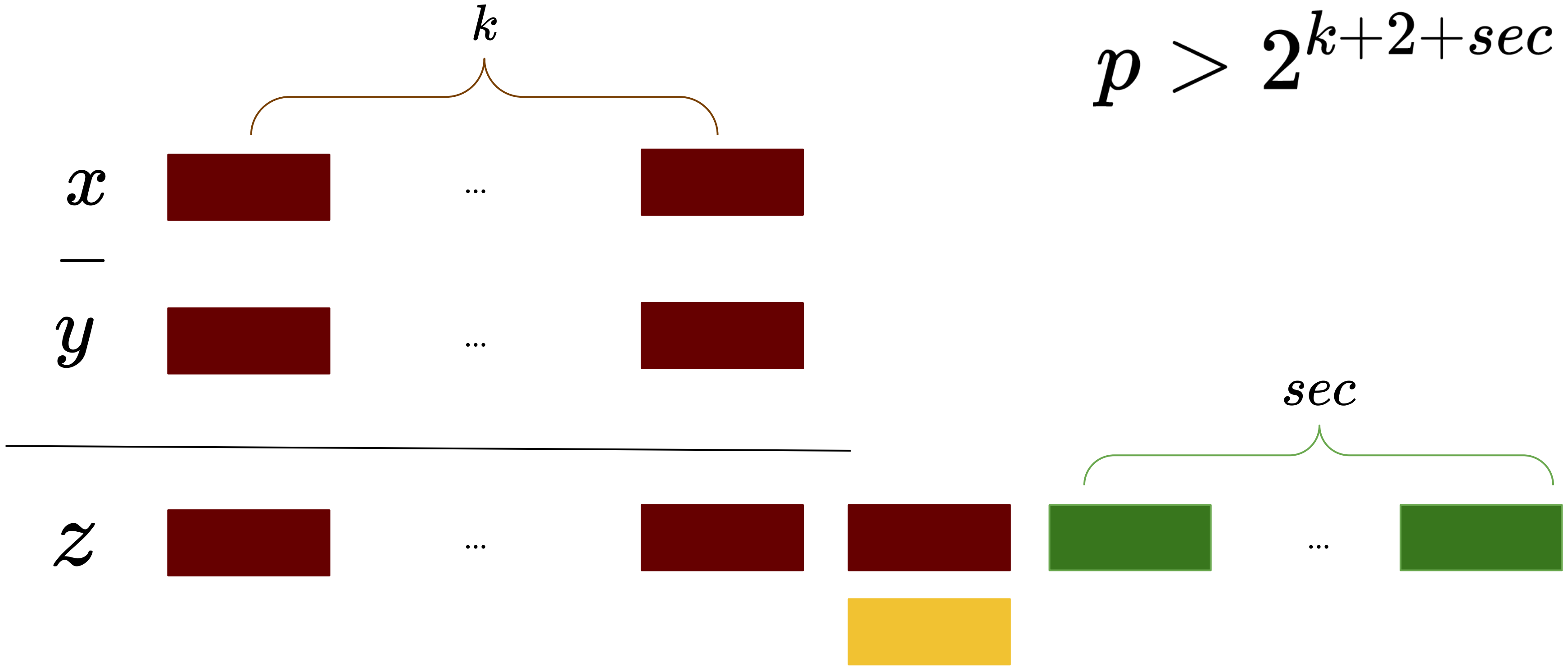
What is a negative number?

$$\begin{aligned} sign^{+} &\in [0, 2^{k-1}) \\ sign^{-} &\in [2^{k-1}, 2^{k}) \end{aligned} \quad \bmod 2^k$$

$$\begin{aligned} sign^{+} &\in [0, p/2) \\ sign^{-} &\in [(p+1)/2, p) \end{aligned} \quad \bmod p$$

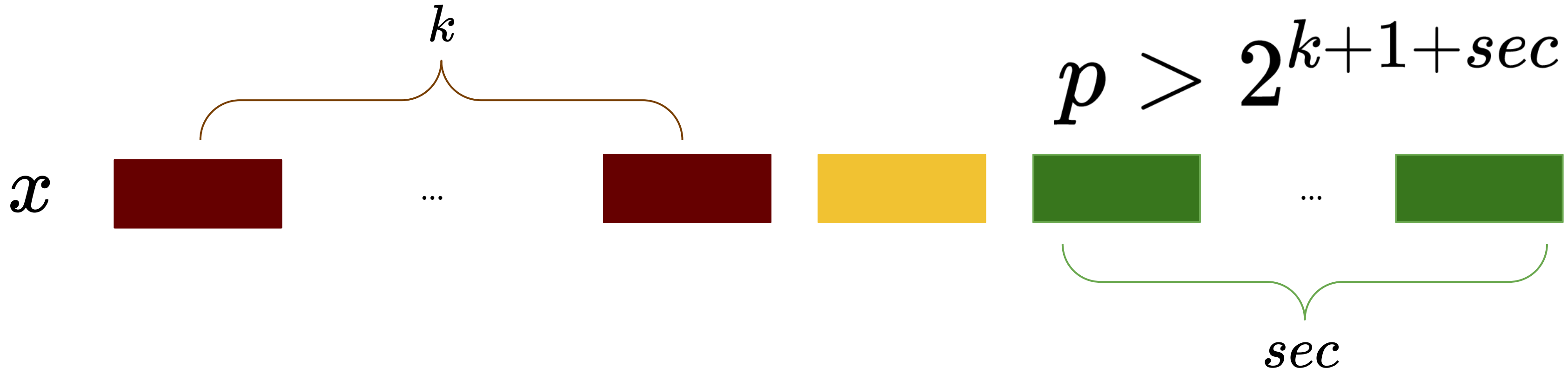


LTS is different to LTC - common approaches



$$p > 2^{k+2+sec}$$

LTS is different to LTC - common approaches



$$p > 2^{k+1+sec}$$

$$x < C$$

LTC - our approach

$$p \approx 2^{x}$$

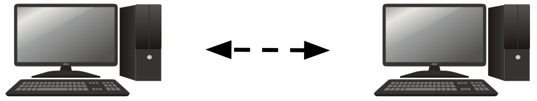
No need for bounding the inputs and faster than previous constructions

$$x < C$$

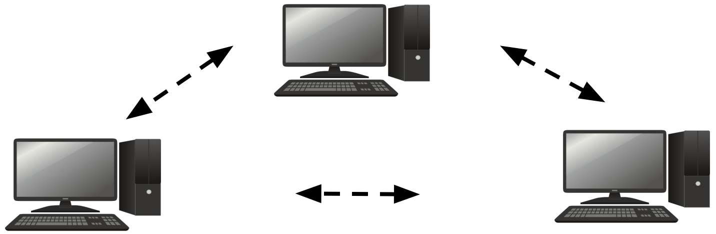
Setup

- Intel(R) Core(TM) i9-9900 CPU @ 3.10GHz
- 128GB of RAM over a 10Gb/s network switch with an average RTT time of 1ms.

Dishonest majority



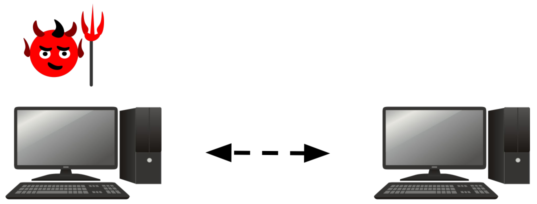
Honest majority



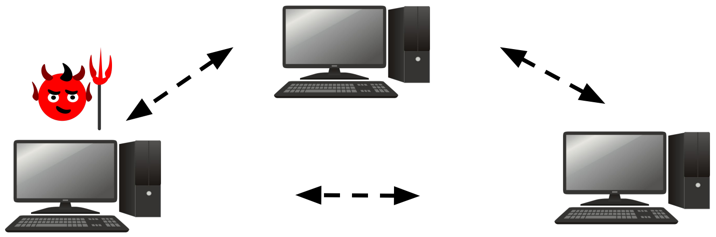
Setup

- Intel(R) Core(TM) i9-9900 CPU @ 3.10GHz
- 128GB of RAM over a 10Gb/s network switch with an average RTT time of 1ms.

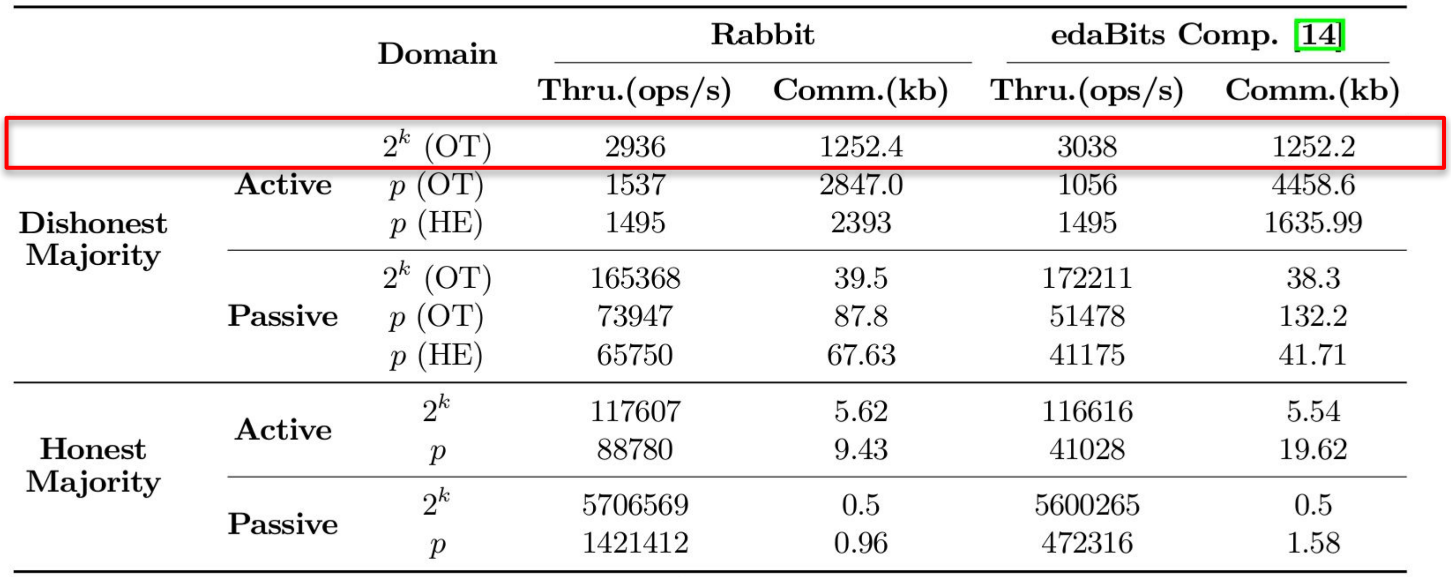
Dishonest majority



Honest majority



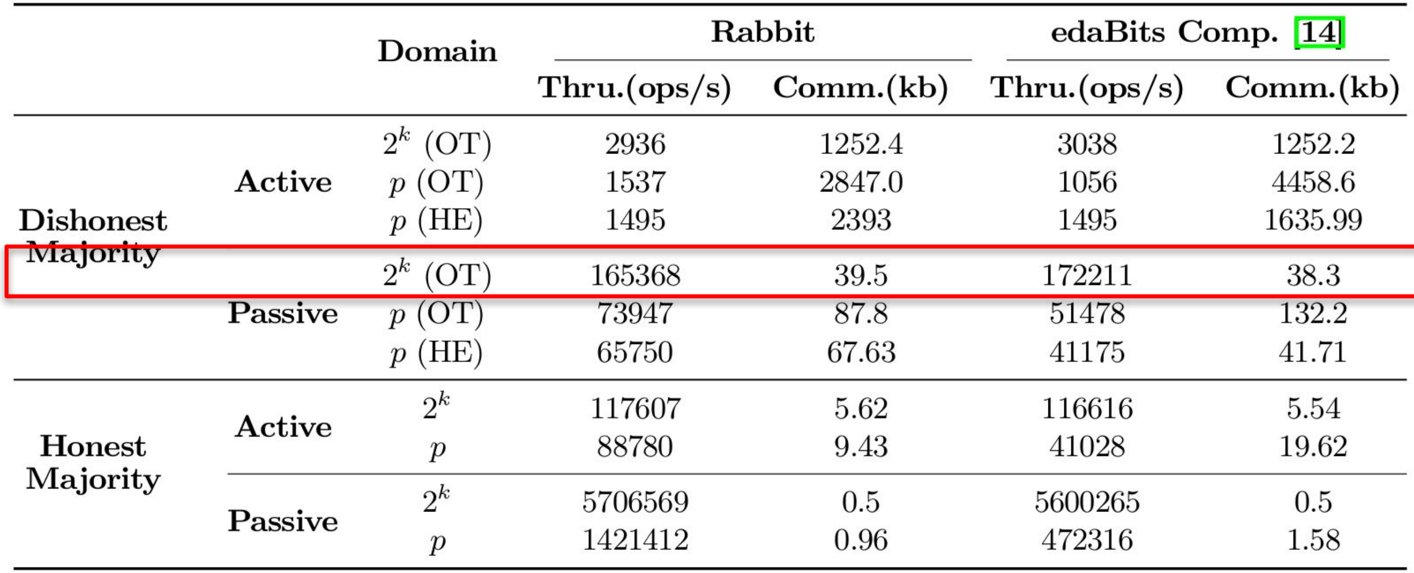
Experiments



		Domain	Rabbit		edaBits Comp. [14]	
			Thru.(ops/s)	**Comm.(kb)**	**Thru.(ops/s)**	**Comm.(kb)**
Dishonest Majority	**Active**	2^k (OT)	2936	1252.4	3038	1252.2
		p (OT)	1537	2847.0	1056	4458.6
		p (HE)	1495	2393	1495	1635.99
	Passive	2^k (OT)	165368	39.5	172211	38.3
		p (OT)	73947	87.8	51478	132.2
		p (HE)	65750	67.63	41175	41.71
Honest Majority	**Active**	2^k	117607	5.62	116616	5.54
		p	88780	9.43	41028	19.62
	Passive	2^k	5706569	0.5	5600265	0.5
		p	1421412	0.96	472316	1.58

Table 1: Throughput and communication for running secure comparisons using Rabbit in contrast to prior art *over LAN* for 16 threads, with 2 million comparisons in total.

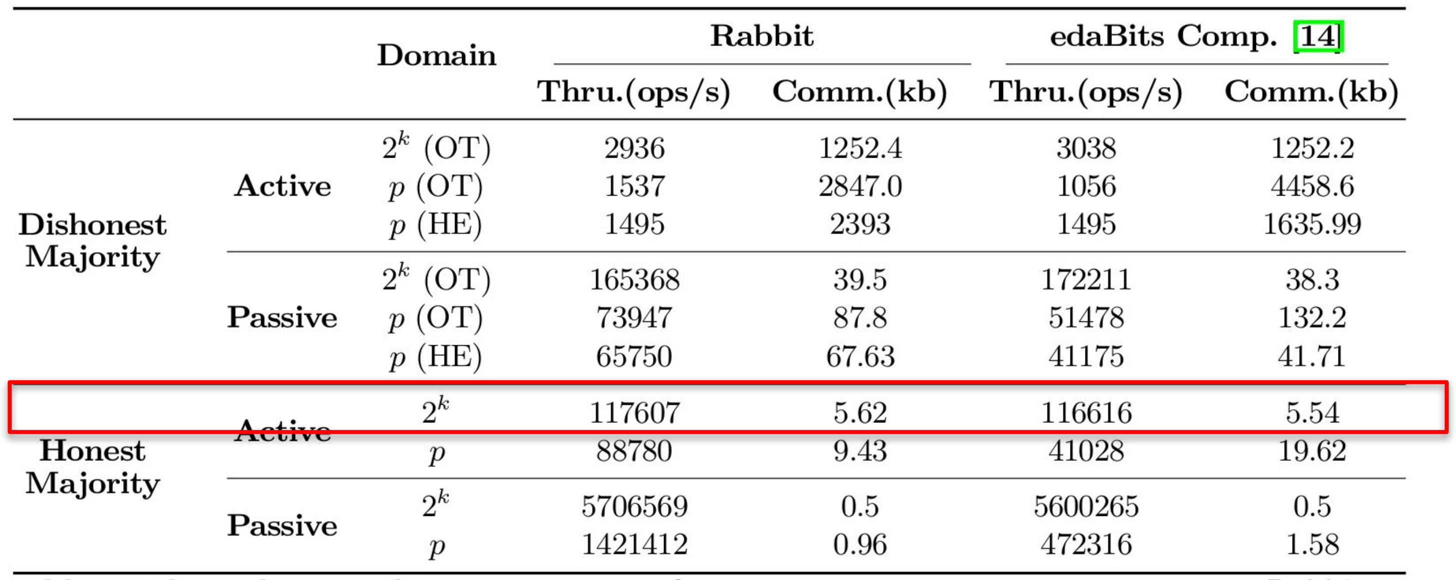
Experiments



		Domain	Rabbit		edaBits Comp. [14]	
			Thru.(ops/s)	**Comm.(kb)**	**Thru.(ops/s)**	**Comm.(kb)**
Dishonest Majority	**Active**	2^k (OT)	2936	1252.4	3038	1252.2
		p (OT)	1537	2847.0	1056	4458.6
		p (HE)	1495	2393	1495	1635.99
	Passive	2^k (OT)	165368	39.5	172211	38.3
		p (OT)	73947	87.8	51478	132.2
		p (HE)	65750	67.63	41175	41.71
Honest Majority	**Active**	2^k	117607	5.62	116616	5.54
		p	88780	9.43	41028	19.62
	Passive	2^k	5706569	0.5	5600265	0.5
		p	1421412	0.96	472316	1.58

Table 1: Throughput and communication for running secure comparisons using Rabbit in contrast to prior art *over LAN* for 16 threads, with 2 million comparisons in total.

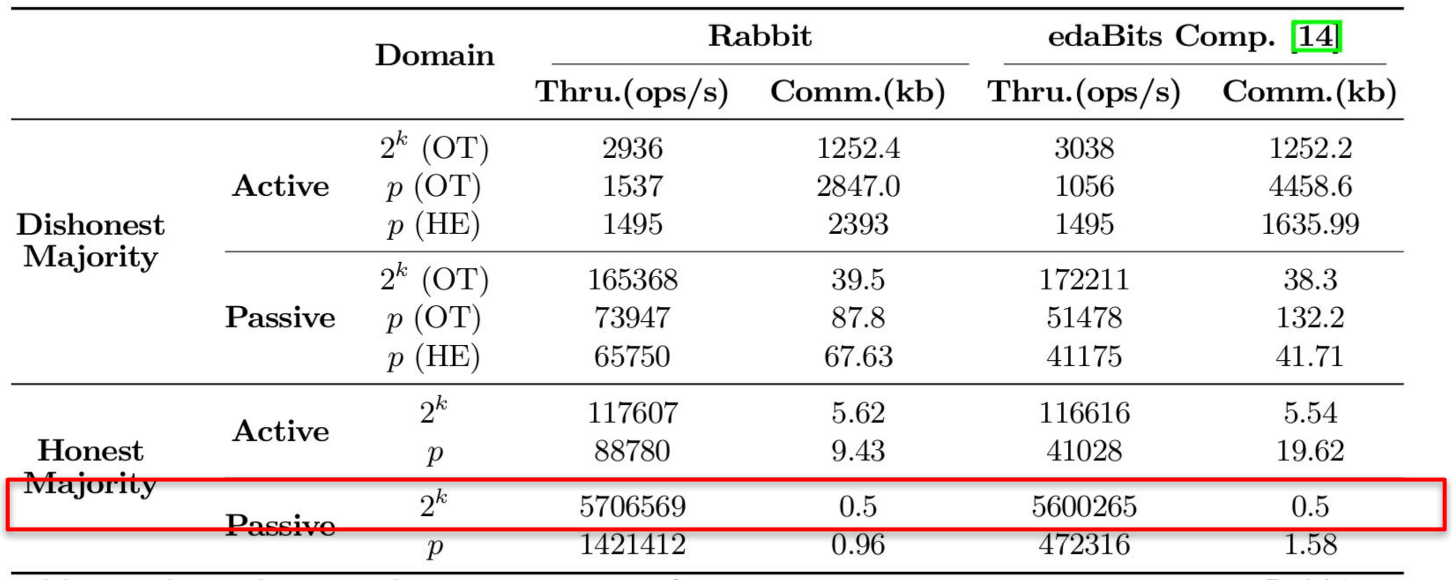
Experiments



		Domain	Rabbit		edaBits Comp. [14]	
			Thru.(ops/s)	Comm.(kb)	Thru.(ops/s)	Comm.(kb)
Dishonest Majority	**Active**	2^k (OT)	2936	1252.4	3038	1252.2
		p (OT)	1537	2847.0	1056	4458.6
		p (HE)	1495	2393	1495	1635.99
	Passive	2^k (OT)	165368	39.5	172211	38.3
		p (OT)	73947	87.8	51478	132.2
		p (HE)	65750	67.63	41175	41.71
Honest Majority	**Active**	2^k	117607	5.62	116616	5.54
		p	88780	9.43	41028	19.62
	Passive	2^k	5706569	0.5	5600265	0.5
		p	1421412	0.96	472316	1.58

Table 1: Throughput and communication for running secure comparisons using Rabbit in contrast to prior art *over LAN* for 16 threads, with 2 million comparisons in total.

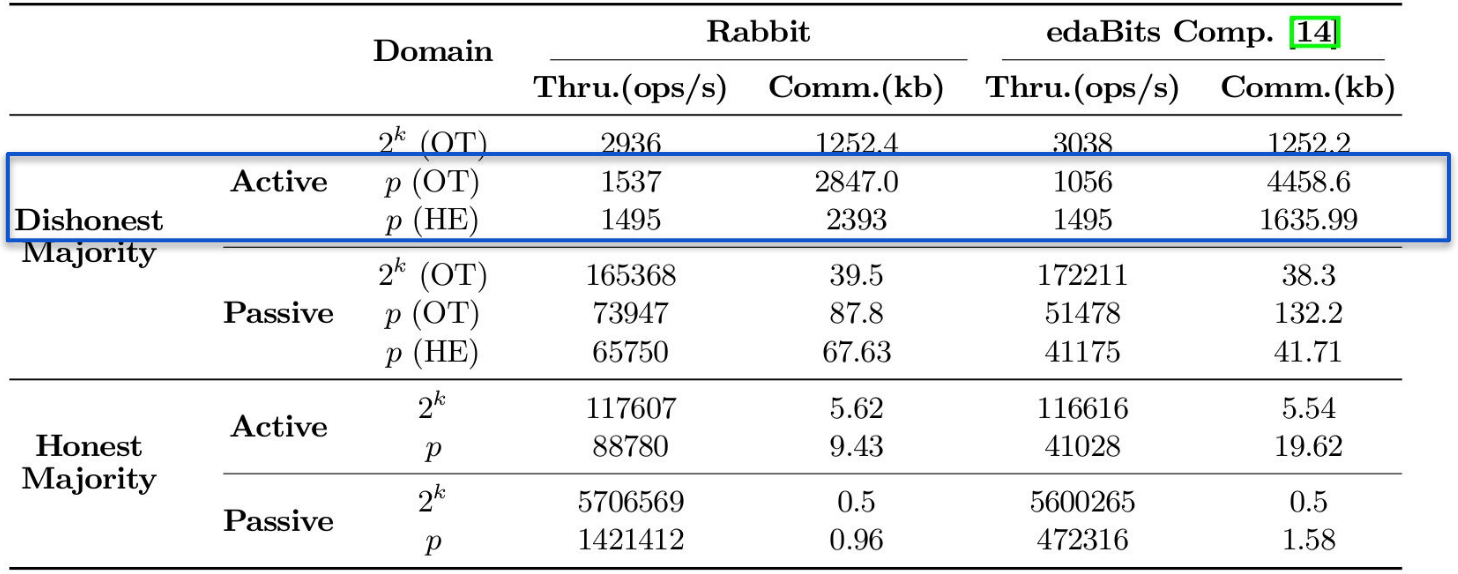
Experiments



		Domain	Rabbit		edaBits Comp. [14]	
			Thru.(ops/s)	Comm.(kb)	Thru.(ops/s)	Comm.(kb)
Dishonest Majority	Active	2^k (OT)	2936	1252.4	3038	1252.2
		p (OT)	1537	2847.0	1056	4458.6
		p (HE)	1495	2393	1495	1635.99
	Passive	2^k (OT)	165368	39.5	172211	38.3
		p (OT)	73947	87.8	51478	132.2
		p (HE)	65750	67.63	41175	41.71
Honest Majority	Active	2^k	117607	5.62	116616	5.54
		p	88780	9.43	41028	19.62
	Passive	2^k	5706569	0.5	5600265	0.5
		p	1421412	0.96	472316	1.58

Table 1: Throughput and communication for running secure comparisons using Rabbit in contrast to prior art *over LAN* for 16 threads, with 2 million comparisons in total.

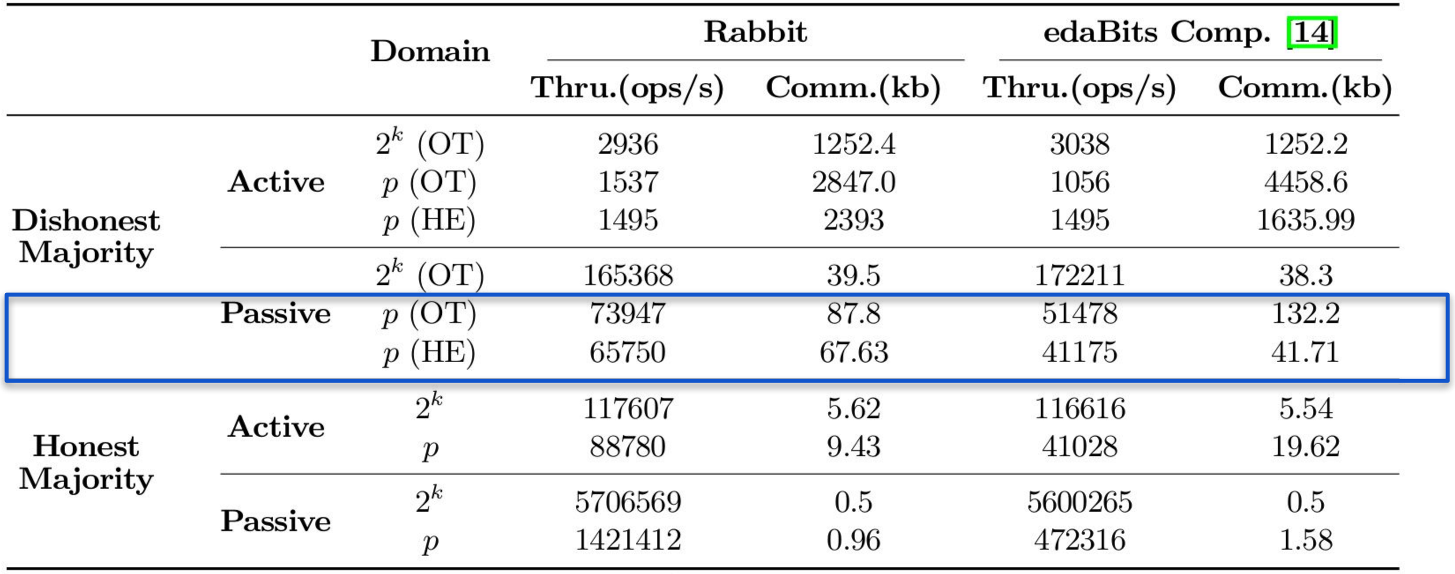
Experiments



		Domain	Rabbit		edaBits Comp. [14]	
			Thru.(ops/s)	Comm.(kb)	Thru.(ops/s)	Comm.(kb)
Dishonest Majority	Active	2^k (OT)	2936	1252.4	3038	1252.2
		p (OT)	1537	2847.0	1056	4458.6
		p (HE)	1495	2393	1495	1635.99
	Passive	2^k (OT)	165368	39.5	172211	38.3
		p (OT)	73947	87.8	51478	132.2
		p (HE)	65750	67.63	41175	41.71
Honest Majority	Active	2^k	117607	5.62	116616	5.54
		p	88780	9.43	41028	19.62
	Passive	2^k	5706569	0.5	5600265	0.5
		p	1421412	0.96	472316	1.58

Table 1: Throughput and communication for running secure comparisons using Rabbit in contrast to prior art *over LAN* for 16 threads, with 2 million comparisons in total.

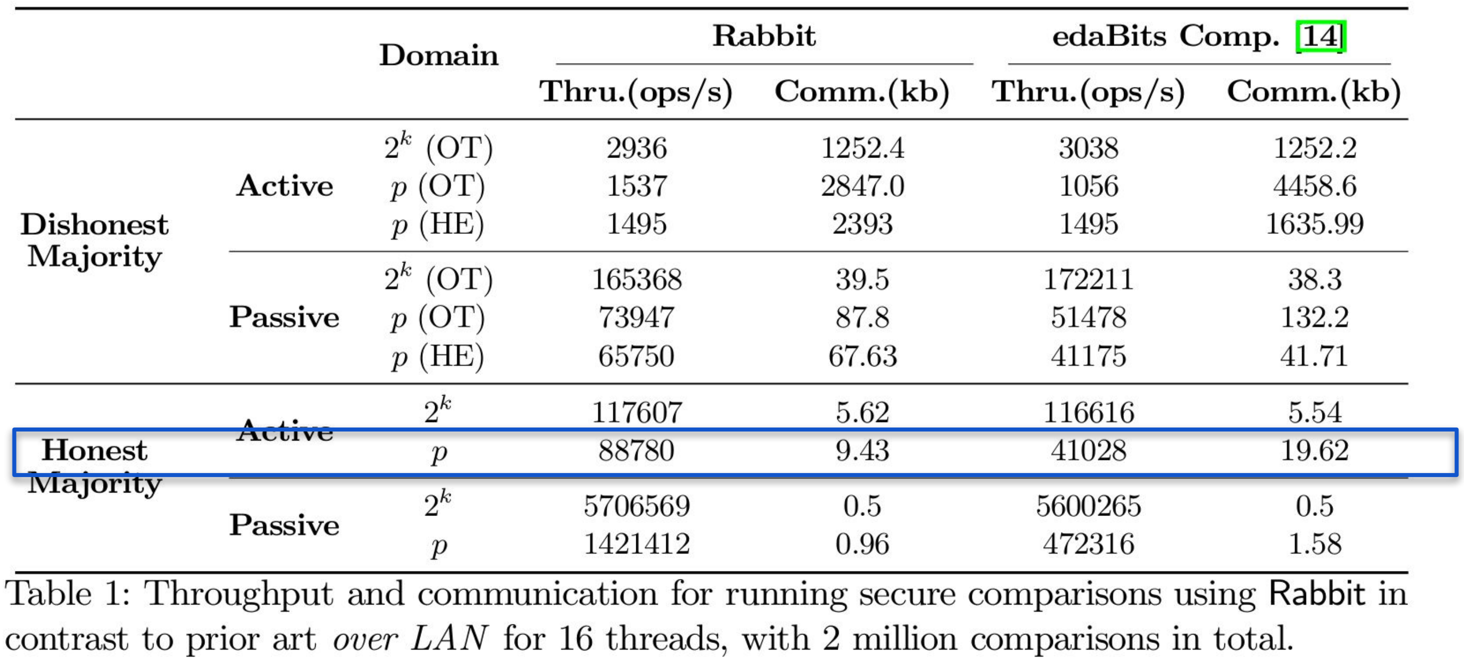
Experiments



		Domain	Rabbit		edaBits Comp. [14]	
			Thru.(ops/s)	Comm.(kb)	Thru.(ops/s)	Comm.(kb)
Dishonest Majority	Active	2^k (OT)	2936	1252.4	3038	1252.2
		p (OT)	1537	2847.0	1056	4458.6
		p (HE)	1495	2393	1495	1635.99
	Passive	2^k (OT)	165368	39.5	172211	38.3
		p (OT)	73947	87.8	51478	132.2
		p (HE)	65750	67.63	41175	41.71
Honest Majority	Active	2^k	117607	5.62	116616	5.54
		p	88780	9.43	41028	19.62
	Passive	2^k	5706569	0.5	5600265	0.5
		p	1421412	0.96	472316	1.58

Table 1: Throughput and communication for running secure comparisons using Rabbit in contrast to prior art *over LAN* for 16 threads, with 2 million comparisons in total.

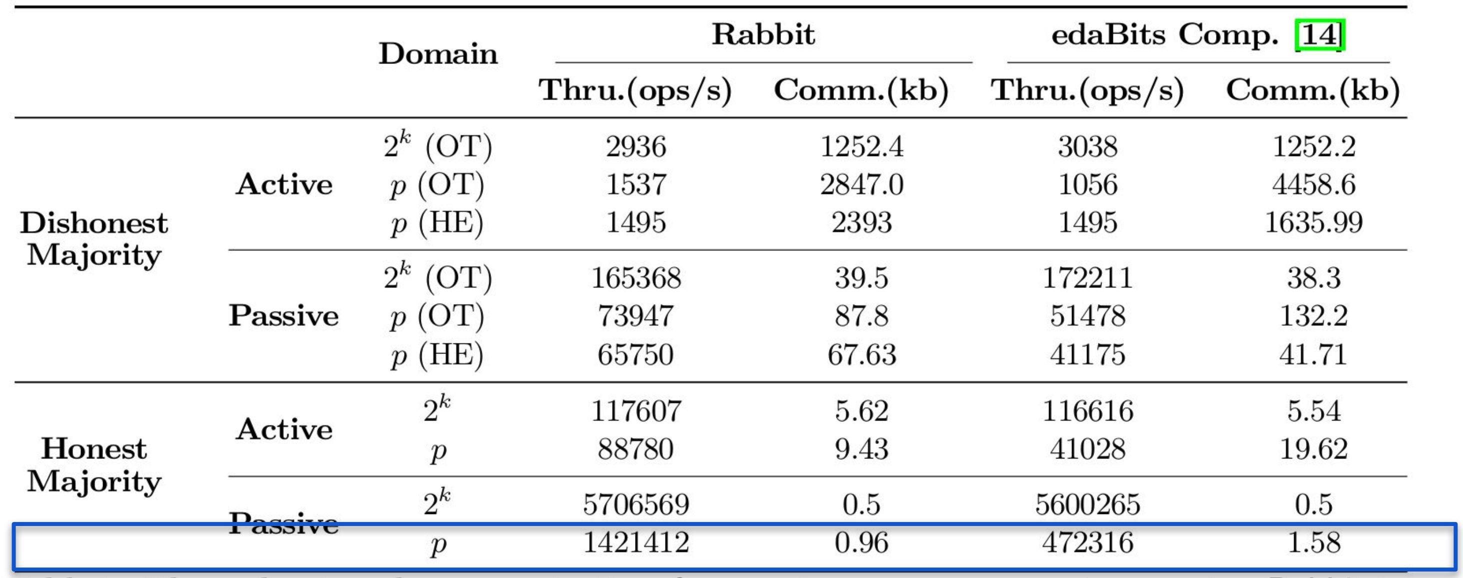
Experiments



		Domain	Rabbit		edaBits Comp. [14]	
			Thru.(ops/s)	Comm.(kb)	Thru.(ops/s)	Comm.(kb)
Dishonest Majority	**Active**	2^k (OT)	2936	1252.4	3038	1252.2
		p (OT)	1537	2847.0	1056	4458.6
		p (HE)	1495	2393	1495	1635.99
	Passive	2^k (OT)	165368	39.5	172211	38.3
		p (OT)	73947	87.8	51478	132.2
		p (HE)	65750	67.63	41175	41.71
Honest Majority	**Active**	2^k	117607	5.62	116616	5.54
		p	88780	9.43	41028	19.62
	Passive	2^k	5706569	0.5	5600265	0.5
		p	1421412	0.96	472316	1.58

Table 1: Throughput and communication for running secure comparisons using Rabbit in contrast to prior art *over LAN* for 16 threads, with 2 million comparisons in total.

Experiments



		Domain	Rabbit		edaBits Comp. [14]	
			Thru.(ops/s)	Comm.(kb)	Thru.(ops/s)	Comm.(kb)
Dishonest Majority	Active	2^k (OT)	2936	1252.4	3038	1252.2
		p (OT)	1537	2847.0	1056	4458.6
		p (HE)	1495	2393	1495	1635.99
	Passive	2^k (OT)	165368	39.5	172211	38.3
		p (OT)	73947	87.8	51478	132.2
		p (HE)	65750	67.63	41175	41.71
Honest Majority	Active	2^k	117607	5.62	116616	5.54
		p	88780	9.43	41028	19.62
	Passive	2^k	5706569	0.5	5600265	0.5
		p	1421412	0.96	472316	1.58

Table 1: Throughput and communication for running secure comparisons using Rabbit in contrast to prior art *over LAN* for 16 threads, with 2 million comparisons in total.

Experiments

For the HE case we noticed that Rabbit has a lower footprint memory due to smaller ciphertexts

Conclusions

- We show that removing the slack can lead to more efficient protocols.
- To get enough statistical security we still need the underlying prime to be smooth. Would be cool to remove this constraint although this was not needed in our applications.

Thank you!

Backup Slides

LTS is different to LTC - common approaches

To compare [x] > [y], assume x and y are k-bit long, compute [x] - [y] and then truncate the output by 2^{k+1} to extract the MSB.

Same approach to compare [x] < C.

Caveat: Most efficient protocols required inputs to be bounded to avoid wraparound of [x] - [y].

We can do better*:

1) reduce the cost of [x] < C
2) eliminate need of slack when comparing [x], [y] or just [x] < C.

** Works with rings or smooth primes (p \approx 2^k) due to edabit form.

For the case of generic secret sharing:

- Probabilistic truncation (MP-SPDZ, SCALE, etc) - relatively fast.
- Deterministic truncation - largest improvement Previous protocols for LTC comparisons using generic secret sharing assumed an input bound.

Brief intro

- MPC what is it ➜ Applications
- What do all these have in common. Comparison. ➜ Overhead is prohibitive?
- Any security model but most improvement in DH
- Our contrib
    - Slack
    - Overhead/dishonest majority/rings and fields?
- Share conversions are important. edaBits

Technical protocols

- Deeper intuition (wrap around) and intuition figure
- Figure for dependence

Discussion section of our protocols

- Slack
- LTS is not the same as LTC
- Experiments
    - Set-up
    - Throughput experiments
    - LAN/WAN
    - Summary?

Functional Dependence of Rabbit

$$\begin{tikzcd}
& \begin{array}{c}\Pi_{\text{LTBits}} \\ \text{Figure 3}\end{array} \arrow[dl] \arrow[dr] & \\
\begin{array}{c}\Pi_{\text{LTS}} \\ \text{Figure 6}\end{array} & & \begin{array}{c}\Pi_{\text{LTC}} \\ \text{Figure 4}\end{array} \arrow[d] \\
& & \begin{array}{c}\Pi_{\text{ReLU}} \\ \text{Figure 5}\end{array}
\end{tikzcd}$$

Comparison of a public constant with a bit decomposed secret
Input : Public constant R and Boolean sharings $[x_0]_2, \cdots, [x_{m-1}]_2$

Comparison of a secret with another secret
Input : Arithmetic sharings $[x]_M$ and $[y]_M$

Comparison of a secret with a constant
Input : Arithmetic sharing $[x]_M$ and public constant R

Specialized comparison for rings with a specific constant
Input : Π_{LTC} when $M = 2^k$ and $R = 2^{k-1}$

Just wanted to add this animation quick since I feel that I pushed you guys onto Google Slides.

You can tweak each element to an appropriate size and similarly make the other equations.

Also, in case you forgot, the addon is called Math Equations. I’ve suppressed slide numbers for now but I think I should be able to fit my stuff in 6 slides (6th is rabbit transition, so you can start at 7).

$$\begin{tikzcd}
{[x]_M} \arrow[r, "+r"] \arrow[d, "+B"] & a = [x+r]_M \arrow[d, "+B"] \\
c = [x+B]_M \arrow[r, "+r"] & b = [x+r+B]_M
\end{tikzcd}$$

Rabbit Intuition: Observation 2

- Exploit the commutativity of addition:

$$\begin{tikzcd}
{[x]_M} \arrow[r, "+r"] \arrow[d, "+B"] & a=[x+r]_M \arrow[d, "+B"] \\
c=[x+B]_M \arrow[r, "+r"] & b=[x+r+B]_M
\end{tikzcd}$$

- Combine with Observation 1:

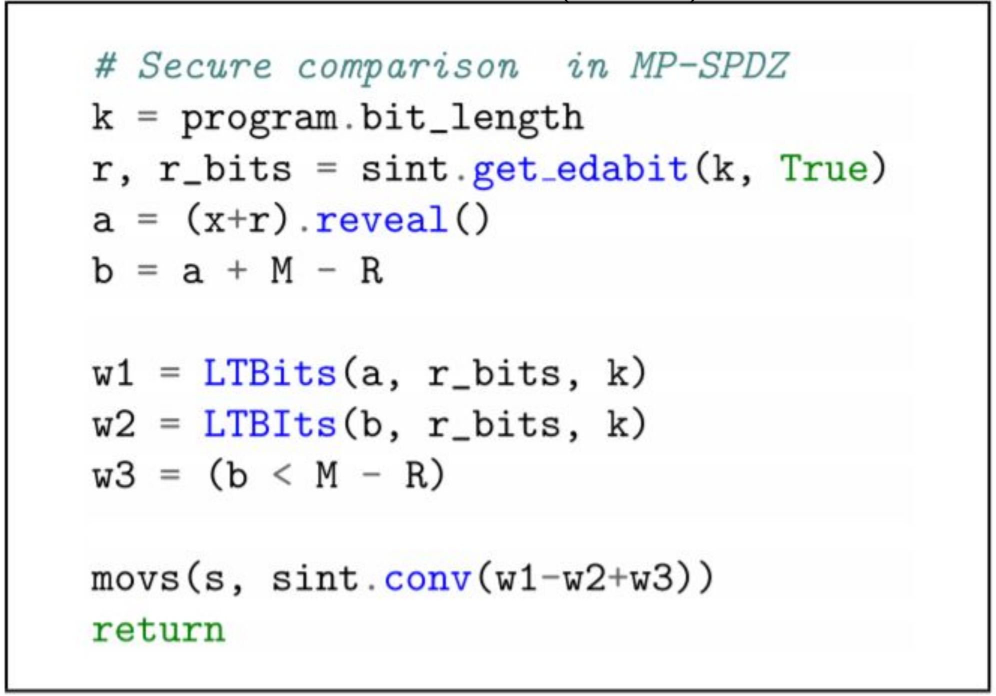
$$\begin{aligned} b &= [a+B] = a+B-M\cdot \mathsf{LT}(b,B) \\ &= x+r-M\cdot \mathsf{LT}(a,r)+B-M\cdot \mathsf{LT}(b,B) \end{aligned}$$

$$\begin{aligned} b &= [c+r] = c+r-M\cdot \mathsf{LT}(b,r) \\ &= x+B-M\cdot \mathsf{LT}(c,B)+r-M\cdot \mathsf{LT}(b,r) \end{aligned}$$

$$\mathsf{LT}(a,r)+\mathsf{LT}(b,B) = \boxed{\mathsf{LT}(c,B)}+\mathsf{LT}(b,r)$$

Rabbit LTC Pseudocode

$$\underset{w_1}{\mathsf{LT}(a,r)} + \underset{w_3}{\mathsf{LT}(b,B)} = \underset{(x \overset{?}{<} R)}{\mathsf{LT}(c,B)} + \underset{w_2}{\mathsf{LT}(b,r)}$$



```
# Secure comparison  in MP-SPDZ
k = program.bit_length
r, r_bits = sint.get_edabit(k, True)
a = (x+r).reveal()
b = a + M - R

w1 = LTBits(a, r_bits, k)
w2 = LTBIts(b, r_bits, k)
w3 = (b < M - R)

movs(s, sint.conv(w1-w2+w3))
return
```