Rerandomizable Garbling, Revisited

Raphael Heitjohann, Jonas von der Heyden, Tibor Jager

IT Security and Cryptography Group
University of Wuppertal

CRYPTO 2025

BERGISCHE UNIVERSITÄT WUPPERTAL

Outsourced Multi-Party Computation

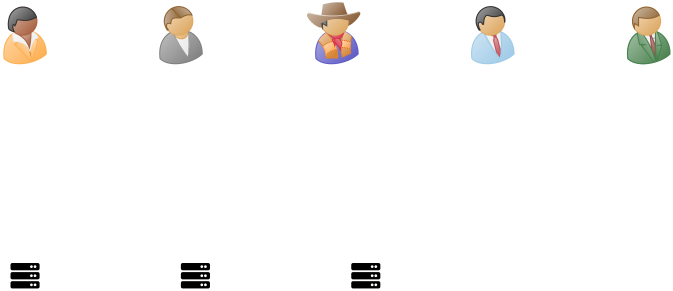
- Setting: large number of resource-constrained clients, small committee of high-performance servers.

Outsourced Multi-Party Computation

- Setting: large number of resource-constrained clients, small committee of high-performance servers.
- Clients outsource MPC to servers without losing privacy.

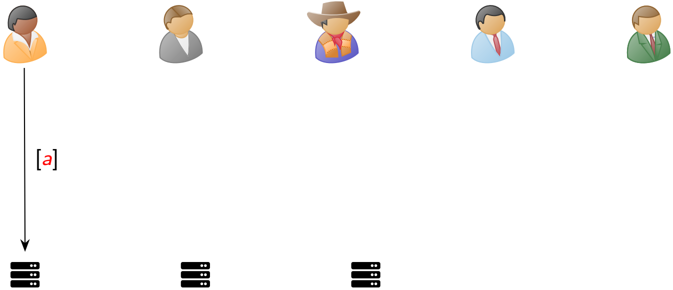
Outsourced Multi-Party Computation

- Setting: large number of resource-constrained clients, small committee of high-performance servers.
- Clients outsource MPC to servers without losing privacy.



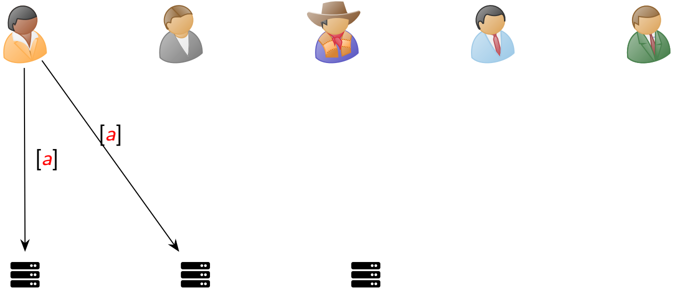
Outsourced Multi-Party Computation

- Setting: large number of resource-constrained clients, small committee of high-performance servers.
- Clients outsource MPC to servers without losing privacy.



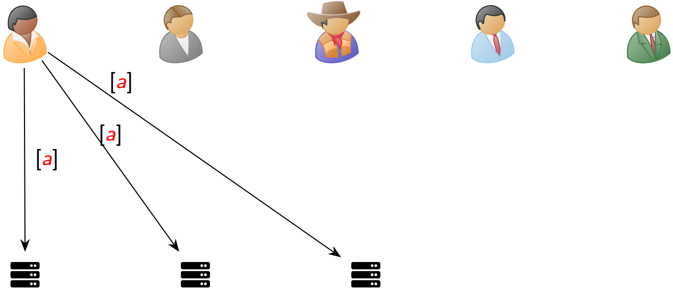
Outsourced Multi-Party Computation

- Setting: large number of resource-constrained clients, small committee of high-performance servers.
- Clients outsource MPC to servers without losing privacy.



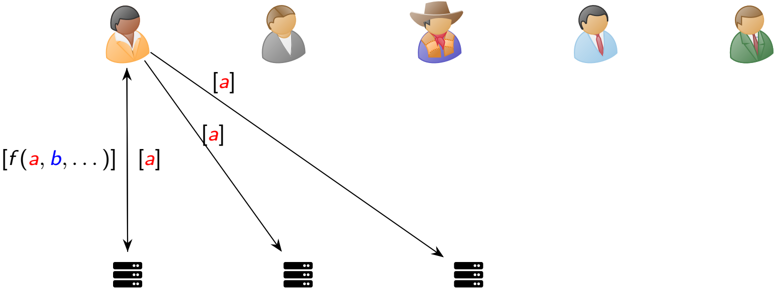
Outsourced Multi-Party Computation

- Setting: large number of resource-constrained clients, small committee of high-performance servers.
- Clients outsource MPC to servers without losing privacy.



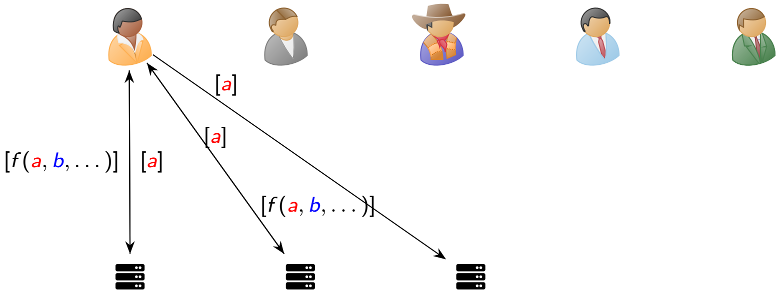
Outsourced Multi-Party Computation

- Setting: large number of resource-constrained clients, small committee of high-performance servers.
- Clients outsource MPC to servers without losing privacy.



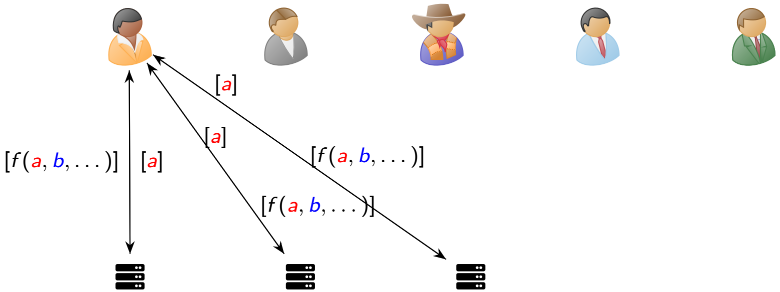
Outsourced Multi-Party Computation

- Setting: large number of resource-constrained clients, small committee of high-performance servers.
- Clients outsource MPC to servers without losing privacy.



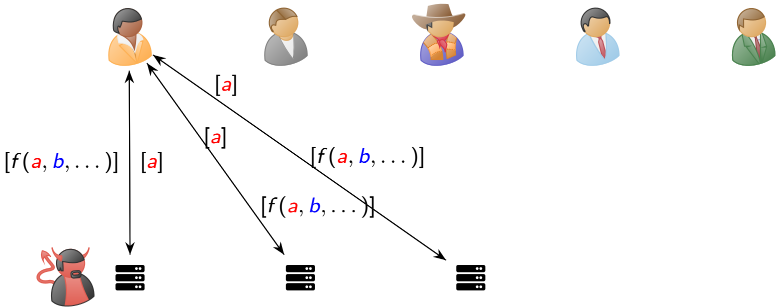
Outsourced Multi-Party Computation

- Setting: large number of resource-constrained clients, small committee of high-performance servers.
- Clients outsource MPC to servers without losing privacy.



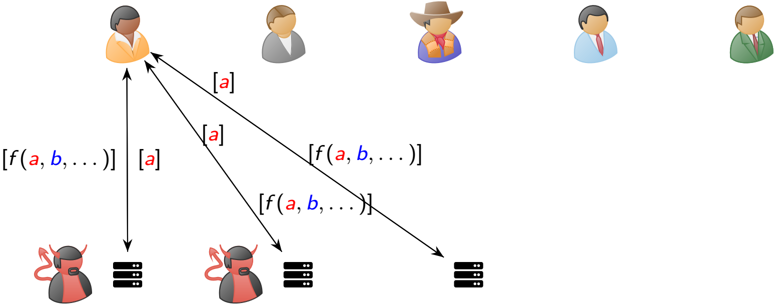
Outsourced Multi-Party Computation

- Setting: large number of resource-constrained clients, small committee of high-performance servers.
- Clients outsource MPC to servers without losing privacy.



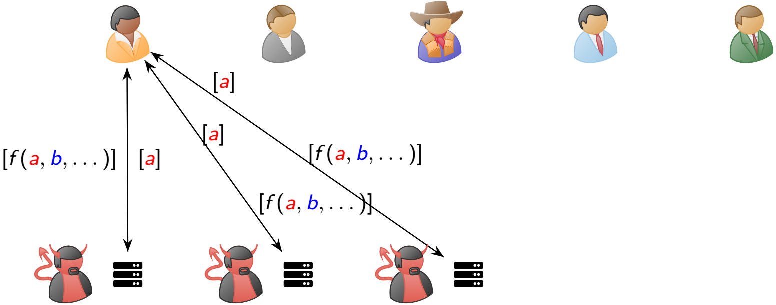
Outsourced Multi-Party Computation

- Setting: large number of resource-constrained clients, small committee of high-performance servers.
- Clients outsource MPC to servers without losing privacy.



Outsourced Multi-Party Computation

- Setting: large number of resource-constrained clients, small committee of high-performance servers.
- Clients outsource MPC to servers without losing privacy.



Outsourced MPC with Semi-Honest Security Against Adaptive Server Corruptions: State of the Art

Feature		**FHE-based** [MTBH21]	**Garbling-based** ("SCALES") [AHKP22]
Adaptive security	**Clients**	✗	✗
	Servers	Only one server required	✓
Constant-round		✓	✓
Corruption threshold t	**Clients**	$t < n$	$t < n$
	Servers	Only one server required	$t < m$
AES (server cost)		∼Minutes	∼Years

Table: Comparison of FHE- and garbling-based outsourced MPC with n clients and m servers.

Outsourced MPC with Semi-Honest Security Against Adaptive Server Corruptions: State of the Art

Feature		**FHE-based** [MTBH21]	**Garbling-based** ("SCALES") [AHKP22]
Adaptive security	**Clients**	✗	✗
	Servers	Only one server required	✓
Constant-round		✓	✓
Corruption threshold t	**Clients**	$t < n$	$t < n$
	Servers	Only one server required	$t < m$
AES (server cost)		$\sim$Minutes	$\sim$Years

Table: Comparison of FHE- and garbling-based outsourced MPC with n clients and m servers.

Outsourced MPC with Semi-Honest Security Against Adaptive Server Corruptions: State of the Art

Feature		**FHE-based** [MTBH21]	**Garbling-based** ("SCALES") [AHKP22]
Adaptive security	**Clients**	✗	✗
	Servers	Only one server required	✓
Constant-round		✓	✓
Corruption threshold t	**Clients**	$t < n$	$t < n$
	Servers	Only one server required	$t < m$
AES (server cost)		∼Minutes	∼Years

Table: Comparison of FHE- and garbling-based outsourced MPC with n clients and m servers.

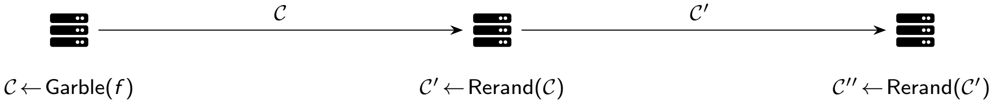
We improve this by 4 orders of magnitude!

Construction of Outsourced MPC via Garbling

- Acharya et al. [AHKP22] construct outsourced MPC scheme "SCALES" based on *rerandomizable garbling schemes* (RGS).

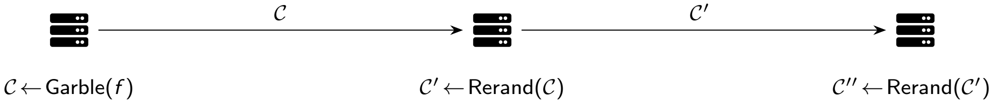
Construction of Outsourced MPC via Garbling

- Acharya et al. [AHKP22] construct outsourced MPC scheme "SCALES" based on *rerandomizable garbling schemes* (RGS).



Construction of Outsourced MPC via Garbling

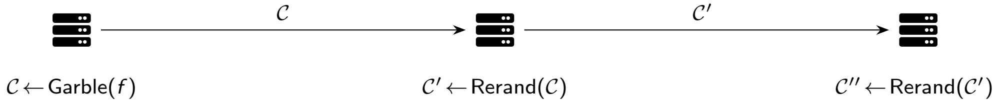
- Acharya et al. [AHKP22] construct outsourced MPC scheme “SCALES” based on *rerandomizable garbling schemes* (RGS).



- RGS require *key-and-message homomorphic encryption* (KMHE).

Construction of Outsourced MPC via Garbling

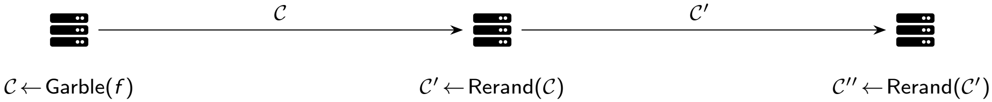
- Acharya et al. [AHKP22] construct outsourced MPC scheme "SCALES" based on *rerandomizable garbling schemes* (RGS).



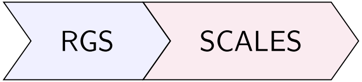
- RGS require *key-and-message homomorphic encryption* (KMHE).

Construction of Outsourced MPC via Garbling

- Acharya et al. [AHKP22] construct outsourced MPC scheme "SCALES" based on *rerandomizable garbling schemes* (RGS).

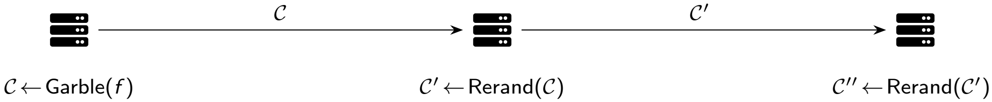


- RGS require *key-and-message homomorphic encryption* (KMHE).



Construction of Outsourced MPC via Garbling

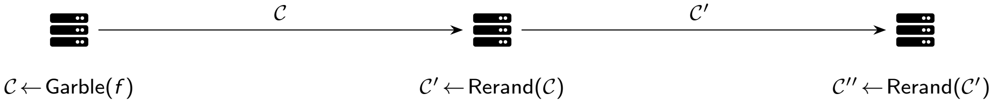
- Acharya et al. [AHKP22] construct outsourced MPC scheme "SCALES" based on *rerandomizable garbling schemes* (RGS).



- RGS require *key-and-message homomorphic encryption* (KMHE).

Construction of Outsourced MPC via Garbling

- Acharya et al. [AHKP22] construct outsourced MPC scheme "SCALES" based on *rerandomizable garbling schemes* (RGS).

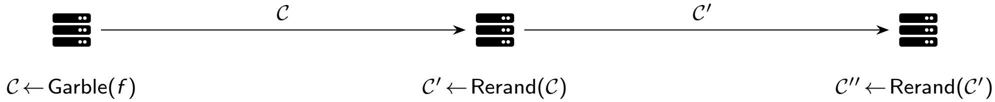


- RGS require *key-and-message homomorphic encryption* (KMHE).

- Only known KMHE so far: [BHHO08] where one encryption holds $2\kappa^2 + 3\kappa + 1$ group elements and requires $\kappa^3 + \kappa^2$ exponentiations.

Construction of Outsourced MPC via Garbling

- Acharya et al. [AHKP22] construct outsourced MPC scheme "SCALES" based on *rerandomizable garbling schemes* (RGS).



- RGS require *key-and-message homomorphic encryption* (KMHE).

- Only known KMHE so far: [BHHO08] where one encryption holds $2\kappa^2 + 3\kappa + 1$ group elements and requires $\kappa^3 + \kappa^2$ exponentiations.

Our Goal

Improve efficiency of the KMHE building block.

Our Contributions

- We construct much more efficient KMHE.

Our Contributions

- We construct much more efficient KMHE.
- We devise new KMHE and RGS definitions that address gaps in [AHKP22] and allow for use of our KMHE.

Our Contributions

- We construct much more efficient KMHE.
- We devise new KMHE and RGS definitions that address gaps in [AHKP22] and allow for use of our KMHE.
- New Gate KMHE primitive allowing for randomness reuse across garbling table ciphertexts.

Performance Estimates for [AHKP22] and our RGS

Table: Comparison of the RGS from [AHKP22] and our work in bytes and clock cycles (cc).

	BHHO-based	Our work	Improvement
Size of one garbled gate	133.43 MB	1.35 MB	98.99 %
Size of one garbled Max circuit	125.87 GB	1.27 GB	98.99 %
Size of one garbled Mult circuit	1.74 TB	18.04 GB	98.99 %
Size of one garbled AES circuit	4.4 TB	45.60 GB	98.99 %
Garbling one gate (cc)	1.28×10^{14} (33 min)	2.44×10^{9} (0.04 s)	99.998 %
Garbling a Max circuit (cc)	1.24×10^{17} (22 d)	2.36×10^{12} (37 s)	99.998 %
Garbling a Mult circuit (cc)	1.75×10^{18} (317 d)	3.33×10^{13} (9 min)	99.998 %
Garbling an AES circuit (cc)	4.43×10^{18} (2 yr)	8.43×10^{13} (20 min)	99.998 %
Rerand. a garbled gate (cc)	5.13×10^{14} (2.23 h)	3.35×10^{9} (0.05 s)	99.9993 %
Rerand. a garbl. Max circuit (cc)	4.95×10^{17} (90 d)	3.24×10^{12} (51 s)	99.9993 %
Rerand. a garbl. Mult circuit (cc)	7.01×10^{18} (3.5 yr)	4.58×10^{13} (11 min)	99.9993 %
Rerand. a garbl. AES circuit (cc)	1.77×10^{19} (9 yr)	1.16×10^{14} (30 min)	99.9993 %

Performance Estimates for [AHKP22] and our RGS

Table: Comparison of the RGS from [AHKP22] and our work in bytes and clock cycles (cc).

	BHHO-based	Our work	Improvement
Size of one garbled gate	133.43 MB	1.35 MB	98.99 %
Size of one garbled Max circuit	125.87 GB	1.27 GB	98.99 %
Size of one garbled Mult circuit	1.74 TB	18.04 GB	98.99 %
Size of one garbled AES circuit	4.4 TB	45.60 GB	98.99 %
Garbling one gate (cc)	1.28×10^{14} (33 min)	2.44×10^{9} (0.04 s)	99.998 %
Garbling a Max circuit (cc)	1.24×10^{17} (22 d)	2.36×10^{12} (37 s)	99.998 %
Garbling a Mult circuit (cc)	1.75×10^{18} (317 d)	3.33×10^{13} (9 min)	99.998 %
Garbling an AES circuit (cc)	4.43×10^{18} (2 yr)	8.43×10^{13} (20 min)	99.998 %
Rerand. a garbled gate (cc)	5.13×10^{14} (2.23 h)	3.35×10^{9} (0.05 s)	99.9993 %
Rerand. a garbl. Max circuit (cc)	4.95×10^{17} (90 d)	3.24×10^{12} (51 s)	99.9993 %
Rerand. a garbl. Mult circuit (cc)	7.01×10^{18} (3.5 yr)	4.58×10^{13} (11 min)	99.9993 %
Rerand. a garbl. AES circuit (cc)	1.77×10^{19} (9 yr)	1.16×10^{14} (30 min)	99.9993 %

Performance Estimates for [AHKP22] and our RGS

Table: Comparison of the RGS from [AHKP22] and our work in bytes and clock cycles (cc).

	BHHO-based	Our work	Improvement
Size of one garbled gate	133.43 MB	1.35 MB	98.99 %
Size of one garbled Max circuit	125.87 GB	1.27 GB	98.99 %
Size of one garbled Mult circuit	1.74 TB	18.04 GB	98.99 %
Size of one garbled AES circuit	4.4 TB	45.60 GB	98.99 %
Garbling one gate (cc)	1.28×10^{14} (33 min)	2.44×10^{9} (0.04 s)	99.998 %
Garbling a Max circuit (cc)	1.24×10^{17} (22 d)	2.36×10^{12} (37 s)	99.998 %
Garbling a Mult circuit (cc)	1.75×10^{18} (317 d)	3.33×10^{13} (9 min)	99.998 %
Garbling an AES circuit (cc)	4.43×10^{18} (2 yr)	8.43×10^{13} (20 min)	99.998 %
Rerand. a garbled gate (cc)	5.13×10^{14} (2.23 h)	3.35×10^{9} (0.05 s)	99.9993 %
Rerand. a garbl. Max circuit (cc)	4.95×10^{17} (90 d)	3.24×10^{12} (51 s)	99.9993 %
Rerand. a garbl. Mult circuit (cc)	7.01×10^{18} (3.5 yr)	4.58×10^{13} (11 min)	99.9993 %
Rerand. a garbl. AES circuit (cc)	1.77×10^{19} (9 yr)	1.16×10^{14} (30 min)	99.9993 %

Key-and-Message Homomorphic Encryption (KMHE)

Requirements for KMHE scheme (intuitive):

Key-and-Message Homomorphic Encryption (KMHE)

Requirements for KMHE scheme (intuitive):

1. Need *same* homomorphism in the key *and* the message space.

Key-and-Message Homomorphic Encryption (KMHE)

Requirements for KMHE scheme (intuitive):

1. Need *same* homomorphism in the key *and* the message space.
   - Not easy, e.g. ElGamal has no (efficiently decryptable) message homomorphism $(\mathbb{Z}_p, +)$.

Key-and-Message Homomorphic Encryption (KMHE)

Requirements for KMHE scheme (intuitive):

1. Need *same* homomorphism in the key *and* the message space.
   - Not easy, e.g. ElGamal has no (efficiently decryptable) message homomorphism $(\mathbb{Z}_p, +)$.
2. Key privacy under leakage: Key k and rerandomized key $\sigma(\mathsf{k})$ do not leak function σ.

Key-and-Message Homomorphic Encryption (KMHE)

Requirements for KMHE scheme (intuitive):

1. Need *same* homomorphism in the key *and* the message space.
   - Not easy, e.g. ElGamal has no (efficiently decryptable) message homomorphism $(\mathbb{Z}_p, +)$.
2. Key privacy under leakage: Key k and rerandomized key $\sigma(\mathsf{k})$ do not leak function σ.
   - E.g. $(\mathbb{Z}_p, +)$ does not work: Key $\mathsf{k} \in \mathbb{Z}_p$ and $\sigma(\mathsf{k}) = \mathsf{k} + \sigma$ leak function σ.

Key-and-Message Homomorphic Encryption (KMHE)

Requirements for KMHE scheme (intuitive):

1. Need *same* homomorphism in the key *and* the message space.
   - Not easy, e.g. ElGamal has no (efficiently decryptable) message homomorphism $(\mathbb{Z}_p, +)$.
2. Key privacy under leakage: Key k and rerandomized key $\sigma(\mathsf{k})$ do not leak function σ.
   - E.g. $(\mathbb{Z}_p, +)$ does not work: Key $\mathsf{k} \in \mathbb{Z}_p$ and $\sigma(\mathsf{k}) = \mathsf{k} + \sigma$ leak function σ.
3. Rerandomization indistinguishable from fresh ciphertext.

Encryption Algorithm of our KMHE (Simplified)

- Setup sets up the bilinear group $\mathsf{pp} = (q, g, h, g_T, \mathbb{G}, \mathbb{H}, \mathbb{G}_T, \mathsf{e})$ and $\mathsf{KGen}(\mathsf{pp})$ samples key $\mathsf{k} \leftarrow\$ \{0,1\}^\kappa$ with Hamming weight $\kappa/2$.

Encryption Algorithm of our KMHE (Simplified)

- Setup sets up the bilinear group $\mathsf{pp} = (q, g, h, g_T, \mathbb{G}, \mathbb{H}, \mathbb{G}_T, \mathsf{e})$ and $\mathsf{KGen}(\mathsf{pp})$ samples key $\mathsf{k} \leftarrow\$ \{0,1\}^\kappa$ with Hamming weight $\kappa/2$.

$\underline{\mathsf{Enc}(\mathsf{pp}, \mathsf{k} \in \{0,1\}^\kappa, \mathsf{m} \in \{0,1\}^\kappa)\text{:}}$

$g_1, \ldots, g_\kappa \leftarrow\$ \mathbb{G}; a_1, \ldots, a_\kappa \leftarrow\$ \mathbb{Z}_q$

$$y \leftarrow \mathsf{e}\left(\prod_{i \in [\kappa]} g_i^{k_i}, h\right)$$

$c_j \leftarrow (h^{a_j}, g_T^{m_j} \cdot y^{a_j}) \quad \forall j \in [\kappa]$

return $\mathsf{ct} \leftarrow (c_1, \ldots, c_\kappa, g_1, \ldots, g_\kappa, y)$

Encryption Algorithm of our KMHE (Simplified)

- Setup sets up the bilinear group $\mathsf{pp} = (q, g, h, g_T, \mathbb{G}, \mathbb{H}, \mathbb{G}_T, \mathsf{e})$ and $\mathsf{KGen}(\mathsf{pp})$ samples key $\mathsf{k} \leftarrow\$ \{0,1\}^\kappa$ with Hamming weight $\kappa/2$.

$\underline{\mathsf{Enc}(\mathsf{pp}, \mathsf{k} \in \{0,1\}^\kappa, \mathsf{m} \in \{0,1\}^\kappa):}$

$g_1, \ldots, g_\kappa \leftarrow\$ \mathbb{G}; a_1, \ldots, a_\kappa \leftarrow\$ \mathbb{Z}_q$

$$y \leftarrow \mathsf{e}\left(\prod_{i \in [\kappa]} g_i^{k_i}, h\right)$$

$c_j \leftarrow (h^{a_j}, g_T^{m_j} \cdot y^{a_j}) \quad \forall j \in [\kappa]$

return $\mathsf{ct} \leftarrow (c_1, \ldots, c_\kappa, g_1, \ldots, g_\kappa, y)$

- We follow [GHV10, AHKP22] by using a bit permutation homomorphism.

Encryption Algorithm of our KMHE (Simplified)

- Setup sets up the bilinear group $\mathsf{pp} = (q, g, h, g_T, \mathbb{G}, \mathbb{H}, \mathbb{G}_T, \mathsf{e})$ and $\mathsf{KGen}(\mathsf{pp})$ samples key $\mathsf{k} \leftarrow\!\$ \{0,1\}^\kappa$ with Hamming weight $\kappa/2$.

$\underline{\mathsf{Enc}(\mathsf{pp}, \mathsf{k} \in \{0,1\}^\kappa, \mathsf{m} \in \{0,1\}^\kappa):}$

$$g_1, \ldots, g_\kappa \leftarrow\!\$ \mathbb{G};\ a_1, \ldots, a_\kappa \leftarrow\!\$ \mathbb{Z}_q$$

$$y \leftarrow \mathsf{e}\left(\prod_{i \in [\kappa]} g_i^{k_i}, h\right)$$

$$c_j \leftarrow (h^{a_j}, g_T^{m_j} \cdot y^{a_j}) \quad \forall j \in [\kappa]$$

$$\text{return } \mathsf{ct} \leftarrow (c_1, \ldots, c_\kappa, g_1, \ldots, g_\kappa, y)$$

- We follow [GHV10, AHKP22] by using a bit permutation homomorphism.
- Key Bit Permutation: Permute $g_1, \ldots, g_\kappa$.

Encryption Algorithm of our KMHE (Simplified)

- Setup sets up the bilinear group $\mathsf{pp} = (q, g, h, g_T, \mathbb{G}, \mathbb{H}, \mathbb{G}_T, \mathsf{e})$ and $\mathsf{KGen}(\mathsf{pp})$ samples key $\mathsf{k} \leftarrow\!\!\$ \{0,1\}^\kappa$ with Hamming weight $\kappa/2$.

$\mathsf{Enc}(\mathsf{pp}, \mathsf{k} \in \{0,1\}^\kappa, \mathsf{m} \in \{0,1\}^\kappa)$:

$g_1, \ldots, g_\kappa \leftarrow\!\!\$ \mathbb{G};\ a_1, \ldots, a_\kappa \leftarrow\!\!\$ \mathbb{Z}_q$

$$y \leftarrow \mathsf{e}\left(\prod_{i \in [\kappa]} g_i^{k_i}, h\right)$$

$c_j \leftarrow (h^{a_j}, g_T^{m_j} \cdot y^{a_j}) \quad \forall j \in [\kappa]$

return $\mathsf{ct} \leftarrow (c_1, \ldots, c_\kappa, g_1, \ldots, g_\kappa, y)$

- We follow [GHV10, AHKP22] by using a bit permutation homomorphism.
- Key Bit Permutation: Permute $g_1, \ldots, g_\kappa$.
- Message Bit Permutation: Permute $c_1, \ldots, c_\kappa$.

Encryption Algorithm of our KMHE (Simplified)

- Setup sets up the bilinear group $\mathsf{pp} = (q, g, h, g_T, \mathbb{G}, \mathbb{H}, \mathbb{G}_T, \mathsf{e})$ and $\mathsf{KGen}(\mathsf{pp})$ samples key $\mathsf{k} \leftarrow\$ \{0,1\}^\kappa$ with Hamming weight $\kappa/2$.

$\mathsf{Enc}(\mathsf{pp}, \mathsf{k} \in \{0,1\}^\kappa, \mathsf{m} \in \{0,1\}^\kappa)$:

$$g_1, \ldots, g_\kappa \leftarrow\$ \mathbb{G};\ a_1, \ldots, a_\kappa \leftarrow\$ \mathbb{Z}_q$$
$$y \leftarrow \mathsf{e}\left(\prod_{i \in [\kappa]} g_i^{k_i}, h\right)$$
$$c_j \leftarrow (h^{a_j}, g_T^{m_j} \cdot y^{a_j}) \quad \forall j \in [\kappa]$$
$$\text{return } \mathsf{ct} \leftarrow (c_1, \ldots, c_\kappa, g_1, \ldots, g_\kappa, y)$$

- We follow [GHV10, AHKP22] by using a bit permutation homomorphism.
- Key Bit Permutation: Permute $g_1, \ldots, g_\kappa$.
- Message Bit Permutation: Permute $c_1, \ldots, c_\kappa$.
- Rerandomization based on DDH.

Encryption Algorithm of our KMHE (Simplified)

- Setup sets up the bilinear group $\mathsf{pp} = (q, g, h, g_T, \mathbb{G}, \mathbb{H}, \mathbb{G}_T, \mathsf{e})$ and $\mathsf{KGen}(\mathsf{pp})$ samples key $\mathsf{k} \leftarrow\$ \{0,1\}^\kappa$ with Hamming weight $\kappa/2$.

$\mathsf{Enc}(\mathsf{pp}, \mathsf{k} \in \{0,1\}^\kappa, \mathsf{m} \in \{0,1\}^\kappa)$:

$g_1, \ldots, g_\kappa \leftarrow\$ \mathbb{G}$; $a_1, \ldots, a_\kappa \leftarrow\$ \mathbb{Z}_q$

$$y \leftarrow \mathsf{e}\left(\prod_{i \in [\kappa]} g_i^{k_i}, h\right)$$

$c_j \leftarrow (h^{a_j}, g_T^{m_j} \cdot y^{a_j}) \quad \forall j \in [\kappa]$

return $\mathsf{ct} \leftarrow (c_1, \ldots, c_\kappa, g_1, \ldots, g_\kappa, y)$

- We follow [GHV10, AHKP22] by using a bit permutation homomorphism.
- Key Bit Permutation: Permute $g_1, \ldots, g_\kappa$.
- Message Bit Permutation: Permute $c_1, \ldots, c_\kappa$.
- Rerandomization based on DDH.

From Quadratic to Linear Size Ciphertexts

The BHHO scheme needed to store every combination $g_i^{a_j}$ (κ^2 many) for decryption, using pairings we can supply the g_i and h^{a_j} separately and then compute their combinations on the fly.

Gate KMHE

Garbling requires some kind of encryption with two keys per ciphertext:

$$
\begin{aligned}
\mathsf{ct}_{00} &= \mathsf{Enc}(\mathsf{k}_0^{(A)}, \mathsf{k}_0^{(B)}, \mathsf{k}_{f(0,0)}) \\
&\vdots \qquad\qquad\qquad \vdots \\
\mathsf{ct}_{11} &= \mathsf{Enc}(\mathsf{k}_1^{(A)}, \mathsf{k}_1^{(B)}, \mathsf{k}_{f(1,1)})
\end{aligned}
$$

Figure: Garbled Table Example.

Gate KMHE

Garbling requires some kind of encryption with two keys per ciphertext:

$$
\begin{aligned}
\mathsf{ct}_{00} &= \mathsf{Enc}(\mathsf{k}_0^{(A)}, \mathsf{k}_0^{(B)}, \mathsf{k}_{f(0,0)}) \\
&\vdots \qquad\qquad\qquad \vdots \\
\mathsf{ct}_{11} &= \mathsf{Enc}(\mathsf{k}_1^{(A)}, \mathsf{k}_1^{(B)}, \mathsf{k}_{f(1,1)})
\end{aligned}
$$

Figure: Garbled Table Example.

Gate KMHE

Gate KMHE primitive encrypts entire garbled table in single operation, halves ciphertext size compared to secret-sharing approach from [AHKP22].

Conclusion

- Rerandomizable garbling allows *adaptively secure*, *outsourced*, and *constant-round* multi-party computation, relying only on a public ledger.

Conclusion

- Rerandomizable garbling allows *adaptively secure*, *outsourced*, and *constant-round* multi-party computation, relying only on a public ledger.
- Our pairing-based KMHE improves ciphertext size by two and runtime by four to five orders of magnitude. Gate KMHE further improves performance by enabling randomness reuse.

Conclusion

- Rerandomizable garbling allows *adaptively secure*, *outsourced*, and *constant-round* multi-party computation, relying only on a public ledger.
- Our pairing-based KMHE improves ciphertext size by two and runtime by four to five orders of magnitude. Gate KMHE further improves performance by enabling randomness reuse.
- New RGS definitions address small gaps in [AHKP22].

Conclusion

- Rerandomizable garbling allows *adaptively secure*, *outsourced*, and *constant-round* multi-party computation, relying only on a public ledger.
- Our pairing-based KMHE improves ciphertext size by two and runtime by four to five orders of magnitude. Gate KMHE further improves performance by enabling randomness reuse.
- New RGS definitions address small gaps in [AHKP22].

Read full version on ePrint:
2025/843

Conclusion

- Rerandomizable garbling allows *adaptively secure*, *outsourced*, and *constant-round* multi-party computation, relying only on a public ledger.
- Our pairing-based KMHE improves ciphertext size by two and runtime by four to five orders of magnitude. Gate KMHE further improves performance by enabling randomness reuse.
- New RGS definitions address small gaps in [AHKP22].

Read full version on ePrint:
2025/843

Hire Jonas!

Conclusion

- Rerandomizable garbling allows *adaptively secure*, *outsourced*, and *constant-round* multi-party computation, relying only on a public ledger.
- Our pairing-based KMHE improves ciphertext size by two and runtime by four to five orders of magnitude. Gate KMHE further improves performance by enabling randomness reuse.
- New RGS definitions address small gaps in [AHKP22].

Read full version on ePrint:
2025/843

Hire Jonas!

Thank you!

Bibliography I

Anasuya Acharya, Carmit Hazay, Vladimir Kolesnikov, and Manoj Prabhakaran.
SCALES - MPC with small clients and larger ephemeral servers.
In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022, Part II*, volume 13748 of *LNCS*, pages 502–531. Springer, Cham, November 2022.

Dan Boneh, Shai Halevi, Michael Hamburg, and Rafail Ostrovsky.
Circular-secure encryption from decision Diffie-Hellman.
In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 108–125. Springer, Berlin, Heidelberg, August 2008.

Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan.
i-Hop homomorphic encryption and rerandomizable Yao circuits.
In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 155–172. Springer, Berlin, Heidelberg, August 2010.

Bibliography II

Yehuda Lindell and Benny Pinkas.
A proof of security of Yao's protocol for two-party computation.
Journal of Cryptology, 22(2):161–188, April 2009.

Christian Mouchet, Juan Ramón Troncoso-Pastoriza, Jean-Philippe Bossuat, and Jean-Pierre Hubaux.
Multiparty homomorphic encryption from ring-learning-with-errors.
PoPETs, 2021(4):291–311, October 2021.

Gate KMHE

- Our (Simplified) KMHE is only a single-key scheme; garbling requires a double-key variant (e.g. [LP09])
- Double-key variant possible via secret-sharing (as in [AHKP22]) or direct construction (a bit more efficient)
- We introduce the Gate KMHE primitive which encrypts the entire garbled table in a single operation, even more efficient

$$\mathsf{ct}_{00} = \mathsf{Enc}(\mathsf{k}_0^{(A)}, \mathsf{k}_0^{(B)}, \mathsf{k}_{f(0,0)})$$

$$\vdots \qquad\qquad \vdots$$

$$\mathsf{ct}_{11} = \mathsf{Enc}(\mathsf{k}_1^{(A)}, \mathsf{k}_1^{(B)}, \mathsf{k}_{f(1,1)})$$

Approach	Size of Garbled Table
Secret-sharing	$8\kappa\ \mathbb{G}, 8\kappa\ \mathbb{H}, 8(\kappa+1)\ \mathbb{G}_T$
Direct double-key	$8\kappa\ \mathbb{G}, 4\kappa\ \mathbb{H}, 4(\kappa+1)\ \mathbb{G}_T$
Gate KMHE	$2\kappa\ \mathbb{G}, 4\kappa\ \mathbb{H}, 4(\kappa+1)\ \mathbb{G}_T$