

“Preview Writeup”: In anticipation of a package submission to the NIST Threshold Call

Title: Symphony: Threshold Evaluation of Symmetric Primitives

Subtitle: A protocol family for threshold AES, SHA2, SHA3 and G-/C-/H-/KMAC evaluation in the three-party, honest majority setting

Version: 0.1 (2026-01-12)¹

Team name: Symphony team

Team members: Hiraku Morita, Erik Pohle, Peter Scholl, Daniel Tschudi

Abstract: This preview writeup describes the package submission Symphony. We intend to submit a protocol family to securely evaluate the AES block cipher, the hash functions SHA2 and SHA3, and all NIST-standardized MAC schemes built from these primitives, i.e., G-/C-/H-/KMAC, in the three-party, honest majority setting with a secret-shared key (where applicable) and secret-shared input. The underlying MPC technique is based on replicated secret sharing over Boolean extension fields, combined with oblivious table lookup protocols. We target active security with abort and the first case of the NIST parameter set for concrete security. Within this package, we will also specify a gadget for preprocessing random one-hot vector correlations, which may be of independent interest. The AES block cipher evaluation will be specified with three protocols for different performance profiles: (i) high online phase throughput, (ii) high total throughput, and (iii) low evaluation latency. Different performance profiles for the threshold hash functions are not planned. Preliminary experimental evaluations indicate a total throughput of ≈ 42000 AES encipherings per second (without key schedule). The reference implementation for threshold hashing and the MAC schemes is ongoing and expected to be completed in the following months.

Proposed crypto-systems:

- (I) OHVerture: gadget for random one-hot vectors (Category S7);
- (II) Maestro: threshold AES encipher/decipher (Category N3.1);
- (III) SHArp: threshold SHA-2 and SHA-3 (Category N3.3);
- (IV) MACnifico: threshold GMAC-AES, CMAC-AES, HMAC and KMAC (Category N3.4);

Keywords: Threshold Cryptography; NIST Threshold Call; Threshold AES; Threshold SHA; Threshold MAC

¹Preliminary version submitted to NIST-MPTC for review

Preview writeup. This document is provided to NIST for online publication, to foster public awareness and support public discussion within the scope of the NIST First Call for Multi-Party Threshold Schemes [NIST-IR8214C]. This “preview writeup” represents a good-faith plan for a subsequent “package submission”. However, until the deadline for package submission, the team may still modify its own composition and the submission plan, including possible changes to the technical scope, and/or the used techniques or achieved results.

Team members: Hiraku Morita ^{i1,a1*}, Erik Pohle ^{i2,a2†}, Peter Scholl ^{i3,a2‡}, Daniel Tschudi ^{i4,a3§,a4#}

Open Researcher and Contributor Identifiers (ORCID):

i1 (0000-0003-3547-7725); i2 (0000-0001-8871-8532); i3 (0000-0002-7937-8422); i4 (0000-0001-6188-1049)

Affiliations:

^{a1} Center for Industrial Software, University of Southern Denmark, Sønderborg, Denmark

^{a2} Dept. of Computer Science, Aarhus University, Aarhus, Denmark

^{a3} Concordium

^{a4} INS, Eastern Switzerland University of Applied Sciences (OST), Rapperswil, Switzerland

Associateship clarifications:

* Assistant Professor. † Postdoctoral researcher; some work performed while at ESAT/COSIC, KU Leuven, Belgium.

‡ Associate Professor. § Senior Researcher. # Lecturer.

Main contacts:

- **Team mailing list:** symphony-submission@googlegroups.com
- **Primary technical contact person:** Erik Pohle, erik.pohle@cs.au.dk
- **Secondary contact person 1:** Peter Scholl, peter.scholl@cs.au.dk
- **Secondary contact person 2:** Hiraku Morita, him@mmmi.sdu.dk
- **Secondary contact person 3:** Daniel Tschudi, dt@concordium.com

Produced by humans. The team hereby confirms that the content in this preview writeup: (i) was produced by the team members, and (ii) was not produced by generative artificial intelligence (AI), with the possible exception of AI-proposed grammar improvements, minor integrated suggestions, or some well-identified and short localized portions of auxiliary content (e.g., some illustration); and (iii) was proofread by the team members.

1. Introduction

Symphony is a family of protocols for threshold evaluation of symmetric, cryptographic primitives in the three-party, honest majority setting. The core MPC protocols extend the actively secure (with abort) replicated secret sharing-based approach by Furukawa et al. [FLNW17] with oblivious table lookup protocols, and are described in Morita et al. [MPSSTT25] (MAESTRO). The planned submission package is based on the code of MAESTRO for the threshold evaluation of AES, and extends the code to support Boolean circuit computation for threshold hash functions and all threshold MAC constructions from NIST Cat. N3.4.

Cryptosystems for Cat. N3. Concretely, we propose cryptosystems for threshold AES enciphering/deciphering (Cat. N3.1), threshold SHA-2 and SHA-3 (Cat. N3.2), and threshold C-/G-/H-/KMAC (Cat. N3.4), where the threshold MAC schemes are built from all threshold block ciphers and hash functions that we support. We describe the exact symmetric-key primitives we plan to support, as well as auxiliary operations in Sect. 2.1.

Gadget for Cat. S7. In addition, the submission package includes the specification of a gadget to generate specific correlated randomness in the form of random one-hot vectors in the same MPC setting (i.e., three-party, honest majority), targeting Cat. S7. We use the gadget internally for efficient block cipher evaluation. Random one-hot vector correlations (over binary fields) have been used in the academic literature to construct efficient MPC protocols for secure sampling from common noise distributions [MRS25; FFGJPWD25], and can in general be used for LUT-based MPC, which extends Boolean circuits with additional table lookup gates. Boolean circuits compiled into LUT gates using hardware synthesis toolchains helps to move communication cost to the preprocessing phase [BHSSY23].

The submission package will contain a full specification of all cryptosystems including a security evaluation, an open-source reference implementation and an experimental performance evaluation.

2. Specification

Here, we give a brief overview of the proposed cryptosystems.

Notation. We write $\mathbb{GF}(2^k)$ to denote the degree- k extension field of \mathbb{F}_2 . We denote a three-party, additive secret sharing of $x \in \mathbb{GF}(2^k)$ as $\langle\langle x \rangle\rangle$, i.e., where $x = \langle\langle x \rangle\rangle_1 + \langle\langle x \rangle\rangle_2 + \langle\langle x \rangle\rangle_3$. Following [FLNW17; MPSSTT25], we write a three-party, replicated secret sharing of $x \in \mathbb{GF}(2^k)$ as $\llbracket x \rrbracket_1 = (\langle\langle x \rangle\rangle_1, \langle\langle x \rangle\rangle_2)$, $\llbracket x \rrbracket_2 = (\langle\langle x \rangle\rangle_2, \langle\langle x \rangle\rangle_3)$, and $\llbracket x \rrbracket_3 = (\langle\langle x \rangle\rangle_3, \langle\langle x \rangle\rangle_1)$. In general, the proposed cryptosystems will work with replicated secret shares, also when secret-shared inputs/outputs are specified.

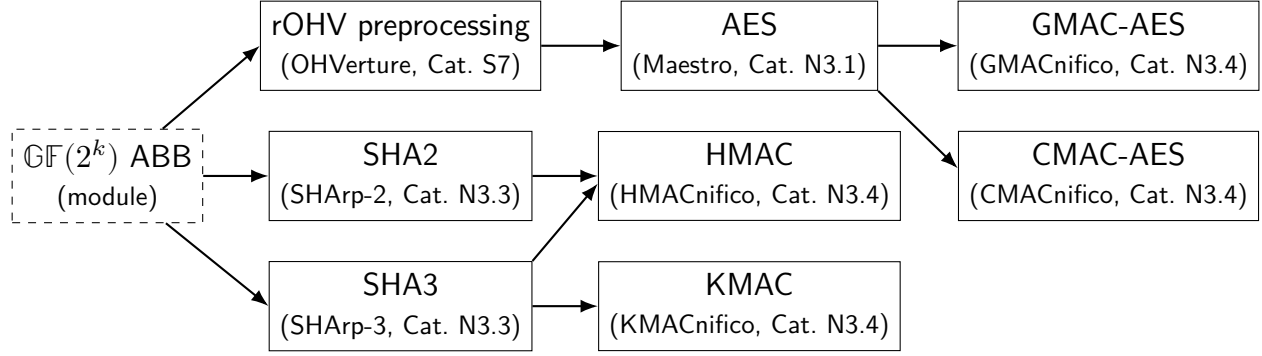


Figure 1: Overview and relationship of the proposed modules and cryptosystems.

2.1. Organization

We propose seven cryptosystems targeting category N3, and one gadget to preprocess correlated randomness for category S7, which we believe is of independent interest. The protocols specified in all proposed cryptosystems make use of an arithmetic black box (ABB) module that realizes MPC for Boolean circuits and Boolean extension field arithmetic. In the following, we briefly describe the interface and proposed functionality of each component. A graphical overview appears in Fig. 1.

We aim to specify the ABB module separately to avoid redundancy and to reduce the complexity of the description of the following proposed cryptosystems. However, since the ABB exposes “unsafe” subprotocols, such as reconstructing a secret-shared value without proper verification of its correctness, its use is internal and we do not recommend using the module to perform arbitrary secure computation.

2.1.1. $\mathbb{GF}(2^k)$ ABB Module

This module realizes the following functions for secure computation of $\mathbb{GF}(2^k)$ -circuits, for $k \in \{1, 4, 8, 64, 128\}$, in the arithmetic black-box style. We propose the operations, for any $x, y \in \mathbb{GF}(2^k)$:

- $\text{Input}(x, P_i) \rightarrow \llbracket x \rrbracket$: party P_i inputs x into the ABB.
- $\text{Linop}(f, \llbracket x_1 \rrbracket, \dots) \rightarrow \llbracket f(x_1, \dots) \rrbracket$, for any $\mathbb{GF}(2)$ -linear function f .
- $\text{LocalMul}(\llbracket x \rrbracket, \llbracket y \rrbracket) \rightarrow \langle\langle x \cdot y \rangle\rangle$
- $\text{Reshare}(\langle\langle x \rangle\rangle) \rightarrow \llbracket x \rrbracket$
- $\text{BitMul}(\llbracket x \rrbracket, \llbracket y \rrbracket) \rightarrow \llbracket x \cdot y \rrbracket$ where $x \in \mathbb{GF}(2)$ and $y \in \mathbb{GF}(2^k)$.
- $\text{Reconst}(\llbracket x \rrbracket) \rightarrow x$: unverified opening of the secret-shared $\llbracket x \rrbracket$.
- $\text{VerifyRecon}()$: verifies all opened values (via Reconst) so far.
- $\text{VerifyMul}()$: verifies all multiplications (via LocalMul and BitMul) so far.
- $\text{Output}(\llbracket x \rrbracket) \rightarrow x/\perp$: verified output

2.1.2. OHVerture: rOHV preprocessing (gadget, Cat. S7)

This gadget outputs correlated randomness in the form of *random one-hot vectors* in two variants. We write $\mathbf{e}^{(r)} \in \mathbb{GF}(2^k)^N$, for $0 \leq r < N$ to denote the vector where $\mathbf{e}^{(r)}[r] = 1$ and $\mathbf{e}^{(r)}[i] = 0$, for all $i \neq r$. The gadget exposes two operations:

- $\text{rOHV}_{\text{rss}}(N = 2^k, L) \rightarrow (\llbracket r_j \rrbracket, \llbracket \mathbf{e}^{(r_j)} \rrbracket)_{j=1}^L$ outputs shares of L correlations, where $r_j \leftarrow \mathbb{GF}(2^k)$. Note that $\mathbf{e}^{(r_j)}$ is shared with replicated secret sharing.
- $\text{rOHV}_{\text{add}}(N = 2^k, L) \rightarrow (\llbracket r_j \rrbracket, \llbracket \langle \mathbf{e}^{(r_j)} \rangle \rrbracket)_{j=1}^L$ outputs shares of L correlations, where $r_j \leftarrow \mathbb{GF}(2^k)$. Note that $\mathbf{e}^{(r_j)}$ is shared with additive secret sharing.

2.1.3. Maestro: AES encipher/decipher (Cat. N3.1)

This cryptosystem evaluates AES- $\{128, 256\}$ with secret-shared key and secret-shared input (optional). We propose the following operations, where $var \in \{128, 256\}$:

- $\text{DistKeyGen}(var) \rightarrow \llbracket \text{ks}^{var} \rrbracket$: outputs the key schedule ks^{var} of a fresh random AES- var key as shares to the parties.
- $\text{InputKey}(k, P_i) \rightarrow \llbracket k \rrbracket$: inputs the key by party P_i .
- $\text{ComputeKS}(\llbracket k \rrbracket, var) \rightarrow \llbracket \text{ks}^{var} \rrbracket$: computes the key schedule.
- $\text{InputBlock}(m, P_i) \rightarrow \llbracket m \rrbracket$: inputs a 128-bit block by party P_i .
- $\text{AESonline}(\llbracket \text{ks}^{var} \rrbracket, \llbracket m \rrbracket) \rightarrow \llbracket \text{AES-}var(\text{ks}, m) \rrbracket$: computes AES enciphering with performance mode “online phase throughput”.
- $\text{AESTp}(\llbracket \text{ks}^{var} \rrbracket, \llbracket m \rrbracket) \rightarrow \llbracket \text{AES-}var(\text{ks}, m) \rrbracket$: computes AES enciphering with performance mode “total throughput”.
- $\text{AESlat}(\llbracket \text{ks}^{var} \rrbracket, \llbracket m \rrbracket) \rightarrow \llbracket \text{AES-}var(\text{ks}, m) \rrbracket$: computes AES enciphering with performance mode “low latency”.
- $\text{OutputBlock}(\llbracket m \rrbracket) \rightarrow m$.

2.1.4. SHArp: Threshold Hashing SHA2/3 (Cat. N3.3)

The cryptosystems SHArp-2 and SHArp-3 evaluate SHA-2 and SHA-3, respectively, with a secret-shared message. We propose the following operations, where $\mathcal{H} \in \{\text{SHA-256}, \text{SHA-512}, \text{SHA3-256}, \text{SHA3-512}\}$:

- $\text{InputMessage}(m, P_i) \rightarrow \llbracket m \rrbracket$: inputs a message by party P_i .
- $\text{Hash-}\mathcal{H}(\llbracket m \rrbracket) \rightarrow \llbracket \mathcal{H}(m) \rrbracket$: computes the threshold hash.
- $\text{OutputDigest}(\llbracket d \rrbracket) \rightarrow d$, where $\llbracket d \rrbracket$ is obtained from any of the \mathcal{H} -computations above.

2.1.5. MACnifico: Threshold GMAC-AES, CMAC-AES, HMAC and KMAC TagGen (Cat. N3.4)

The cryptosystems G-/CMACnifico evaluate G-/CMAC-AES, and H-/KMACnifico evaluate H-/KMAC. We propose the following operations:

- $\text{InputKey}(k, P_i) \rightarrow \llbracket k \rrbracket$: inputs the MAC key by party P_i .
- $\text{InputMessage}(m, P_i) \rightarrow \llbracket m \rrbracket$: inputs the message by party P_i .
- $\text{TagGen}(\llbracket k \rrbracket, \llbracket m \rrbracket) \rightarrow \llbracket \tau \rrbracket$: computes the threshold MAC tag generation on the given key and message. Supported algorithms are GMAC-AES- $\{128, 256\}$, CMAC-AES- $\{128, 256\}$, HMAC- $\{\text{SHA-256}, \text{SHA-512}, \text{SHA3-256}, \text{SHA3-512}\}$ and KMAC- $\{128, 256\}$.
- $\text{OutputTag}(\llbracket \tau \rrbracket) \rightarrow \tau$, where $\llbracket \tau \rrbracket$ is obtained from any thresholdized TagGen computation.

Remark on Input and Output. For the cryptosystems Maestro, SHArp and MACnifico, we specify plaintext input (of a key, block or message) from a MPC or external party, or secret-shared input from a MPC party. Likewise, output (of an enciphered block, digest or tag) is either specified as secret-shared output or plaintext output to a MPC or external party.

2.2. System Model

Our proposed cryptosystems are specified for three-party computation in the honest majority setting, i.e., one corrupted party is tolerated, with static malicious corruption and security with abort. While security is analysed in the synchronous network model, our implementation runs over TCP to setup authenticated point-to-point channels secured with TLS1.3, where session keys can either be derived from pre-shared secrets or negotiated using a PKI. Only the secure channels require a trusted setup, any other MPC-related secrets are securely generated in a distributed way.

Implications. When expecting a message from a party, other parties wait up to a specified timeout and abort the protocol if the timeout is reached. When the TLS/TCP/IP connection raises errors, the party will abort the protocol. We do not provide guaranteed output delivery but if one party aborts, all other parties following the protocol will also abort.

2.3. Security

Our proposal targets the first case of the NIST parameter set for security, i.e., classic computational security of ≥ 128 bit and statistical security of ≥ 40 bit. There are no theoretical limitations that prevent an implementation of the second case of the NIST parameter set. All proposed cryptosystems are built from information-theoretic MPC protocols that make use of well-studied symmetric-key primitives (e.g., CSPRNGs, hash functions) to save communication. As such, our cryptosystems are plausibly post-quantum secure with the exception of the TLS secure channel, which can be (*independently to our protocol*) upgraded to PQ-security as soon as appropriate software libraries become widely available and recommended.

Formal Security Analysis. The proposed cryptosystems are analysed in the synchronous network model (round-based) in the universal composability model. Proofs concerning the security of the $\mathbb{GF}(2^k)$ ABB module, OHVerture and Maestro can be found in the full version [MPSSTT24] of [MPSSTT25]. Proofs for the remaining systems will be delivered in the submission package.

3. Open-Source Implementation

Our implementation uses parts of the code of [MPSSTT25], and extends it for threshold hashing and threshold MAC tag generation. The core code is written in the Rust programming language and uses cargo to build, test and manage dependencies.

Code structure. The core code is structured into multiple crates. Each proposed cryptosystem is implemented in its own crate. In addition, networking, thread management and the arithmetic black box module are implemented in another crate.

Code progress and availability. The core code will be made available in a public git repository under the control of Aarhus University. We are currently extracting relevant code from [MPSSTT25], which already implements the ABB module, rOHV preprocessing (OHVerture) and AES (Maestro), and restructuring it, where appropriate. This code is already well tested, comes with benchmark scripts and instructions. Parties interested in early public testing may use the code of [MPSSTT25], available on github². The code for threshold hashing and threshold MACs will be developed in the coming months. Since the major complexity (ABB and block cipher evaluation) is already implemented in MAESTRO, we expect a smooth development progress.

Implementation of the networking model. We use TLS over TCP/IP to establish authenticated, secure and reliable point-to-point channels between the parties. Our protocols and the security setting do not require broadcast.

Testing. Our code makes heavy use of cargo’s unit testing framework to ensure correctness and the criterion crate to automate benchmarking. All used randomness is seeded for reproducibility. We currently only test malicious behaviour via unit tests by altering the opened message (commitment) or transcript (compare-view) in code related to setting up joint randomness and verifying reconstructed shares. Failing to send messages or sending messages with incorrect syntax (e.g., too short/too long, invalid encoding, etc.) is currently not covered in our tests. We aim to make our code base “robust” in the sense that these conditions cause a program crash (and thus abort).

4. Experimental Performance Evaluation

We report preliminary experimental benchmark results of AES enciphering (Maestro) without key schedule, which uses both the $\mathbb{GF}(2^k)$ ABB and OHVerture, obtained from the code of [MPSSTT25].

Comparing Communication Cost of Maestro. Evaluating a Boolean circuit with recent, communication-efficient three-party protocols, e.g., [LDHHZS23], requires 5120 bits to be sent in 60 rounds. The round complexity can be halved by combining the $\mathbb{GF}(2^8)$ representation by Chida et al. [CHIKP18] with [BGIN19] for active security. This approach still sends 5120 bits. Alternatively, a constant-round implementation based on garbled circuits in the three-party, honest majority setting [RR21; MRZ15] requires $\times 120$ more communication. The AES_{tp} enciphering requires 30 rounds and sends 3200 bits. AES_{online} uses 20 rounds with 2560 bits in the online phase and 2 rounds with 1760 bits of communication in the preprocessing phase. Finally, AES_{lat}

²<https://github.com/KULeuven-COSIC/maestro>

Table 1: Runtime and measured communication cost of Maestro for a batched enciphering of 100 000 AES blocks without key schedule

Protocol	Preprocessing		Online Phase		Throughput (blocks/s)	
	Time (s)	Data (MB)	Time (s)	Data (MB)	Online	Total
AES _{Stp} (Maestro)	–	–	2.34	≈ 40	42799	42799
AES _{Online} (Maestro)	0.23	22	2.24	≈ 32	44624	40533
AES _{lat} (Maestro)	0.84	44	3.65	≈ 32	27373	22280

aims for low-latency and only uses 10 rounds in the online phase with 2560 bits at the cost of 3520 bits of communication in 2 rounds in the preprocessing. Tables 1, 5, 6, 7 and 8 in [MPSSTT24] contain further details about the trade-offs.

Performance. We reproduce some benchmark results from [MPSSTT25] in Table 1, where each party runs on a separate server (16-core Intel Core i9-9900 3.10GHz, 128GB RAM) over a 9.47 Gbit/s LAN network in a multi-threaded implementation.

Platform. We do not anticipate problems using the baseline platform. The multiplication verification protocol in the $\mathbb{GF}(2^k)$ ABB module makes use of carry-less multiplication instructions (to accelerate $\mathbb{GF}(2^{64})$ arithmetic). Our implementation uses intrinsics for Arm Neon and Intel x86_64 SSE2 instruction sets. A plain software fallback implementation exists but is not recommended. We expect the baseline platform to offer one of these instruction sets. A more suitable platform for benchmark evaluation would be three separate machines connected via a low-latency network.

5. Licensing, Patent Claims, and Funding

Licensing. The core code is licensed under MIT license. All dependencies are publicly available rust libraries (called crates in the rust ecosystem). We do not plan to bundle dependencies as they can be conveniently downloaded, compiled and linked using the official Rust package manager cargo. The dependencies and versions are listed in the Cargo.toml files of the respective crates that implement each individual cryptosystem. All direct dependencies can be licensed under MIT license.

Patent Claims. The authors are not aware of any patent claims that may cover the contents of the submission.

Funding Acknowledgement. Large parts of work on the $\mathbb{GF}(2^k)$ ABB module, OHVerture and Maestro were funded by the Flemish Government through FWO SBO project MOZAIK S003321N, by CyberSecurity Research Flanders with reference number VR20192203, by the Danish Independent Research Council under Grant-ID DFF-0165-00107B (C3PO), by the Digital Research Center Denmark (DIREC), by the DIGIT Aarhus University Centre for Digitalisation, Big Data and Data Analytics, by JSPS KAKENHI Grant Number JP21H05052, and by JST CREST Grant Number JPMJCR22M1 (in the scope of [MPSSTT25]). Further development and the submission will partly be funded by a junior postdoctoral fellowship 12A1F26N from the Research Foundation Flanders (FWO), and by DIREC, NFC, and Industriens Fond, SECUREAM project (ID: 25808).

References

- [BGIN19] Elette Boyle, Niv Gilboa, Yuval Ishai, and Ariel Nof. “Practical Fully Secure Three-Party Computation via Sublinear Distributed Zero-Knowledge Proofs”. In: *ACM CCS 2019: 26th Conference on Computer and Communications Security*. Ed. by Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz. London, UK: ACM Press, November 2019, pp. 869–886. DOI: [10.1145/3319535.3363227](https://doi.org/10.1145/3319535.3363227). Also at ia.cr/2019/1390.
- [BHSSY23] Andreas Brüggemann, Robin Hundt, Thomas Schneider, Ajith Suresh, and Hossein Yalame. “FLUTE: Fast and Secure Lookup Table Evaluations”. In: *2023 IEEE Symposium on Security and Privacy*. San Francisco, CA, USA: IEEE Computer Society Press, May 2023, pp. 515–533. DOI: [10.1109/SP46215.2023.10179345](https://doi.org/10.1109/SP46215.2023.10179345). Also at ia.cr/2023/499.
- [CHIKP18] Koji Chida, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, and Benny Pinkas. “High-Throughput Secure AES Computation”. In: *Proceedings of the 6th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography*. WAHC’18. 2018, pp. 13–24. DOI: [10.1145/3267973.3267977](https://doi.org/10.1145/3267973.3267977).
- [FFGJPWD25] Olive Franzese, Congyu Fang, Radhika Garg, Somesh Jha, Nicolas Papernot, Xiao Wang, and Adam Dziedzic. *Secure Noise Sampling for Differentially Private Collaborative Learning*. Cryptology ePrint Archive, Report 2025/1025. 2025. URL: <https://eprint.iacr.org/2025/1025>. Published in CCS’25.
- [FLNW17] Jun Furukawa, Yehuda Lindell, Ariel Nof, and Or Weinstein. “High-Throughput Secure Three-Party Computation for Malicious Adversaries and an Honest Majority”. In: *Advances in Cryptology – EUROCRYPT 2017, Part II*. Ed. by Jean-Sébastien Coron and Jesper Buus Nielsen. Vol. 10211. Lecture Notes in Computer Science. Paris, France: Springer, Cham, Switzerland, April 2017, pp. 225–255. DOI: [10.1007/978-3-319-56614-6_8](https://doi.org/10.1007/978-3-319-56614-6_8). Also at ia.cr/2016/944.
- [LDHHZS23] Yun Li, Yufei Duan, Zhicong Huang, Cheng Hong, Chao Zhang, and Yifan Song. “Efficient 3PC for Binary Circuits with Application to Maliciously-Secure DNN Inference”. In: *USENIX Security 2023: 32nd USENIX Security Symposium*. Ed. by Joseph A. Calandrino and Carmela Troncoso. Anaheim, CA, USA: USENIX Association, August 2023, pp. 5377–5394. URL: <https://www.usenix.org/conference/usenixsecurity23/presentation/li-yun>.
- [MPSSTT24] Hiraku Morita, Erik Pohle, Kunihiko Sadakane, Peter Scholl, Kazunari Tozawa, and Daniel Tschudi. *MAESTRO: Multi-party AES using Lookup Tables*. Cryptology ePrint Archive, Report 2024/1317. 2024. URL: <https://eprint.iacr.org/2024/1317>.
- [MPSSTT25] Hiraku Morita, Erik Pohle, Kunihiko Sadakane, Peter Scholl, Kazunari Tozawa, and Daniel Tschudi. “MAESTRO: Multi-party AES using Lookup Tables”. In: *USENIX*

- Security 2025: 34th USENIX Security Symposium*. Ed. by Lujó Bauer and Giancarlo Pellegrino. Seattle, WA, USA: USENIX Association, August 2025. URL: <https://www.usenix.org/conference/usenixsecurity25/presentation/morita>.
- [MRS25] Fredrik Meisingseth, Christian Rechberger, and Fabian Schmid. *Accelerating Multiparty Noise Generation Using Lookups*. Cryptology ePrint Archive, Paper 2025/805. 2025. URL: <https://eprint.iacr.org/2025/805>.
- [MRZ15] Payman Mohassel, Mike Rosulek, and Ye Zhang. “Fast and Secure Three-party Computation: The Garbled Circuit Approach”. In: *ACM CCS 2015: 22nd Conference on Computer and Communications Security*. Ed. by Indrajit Ray, Ninghui Li, and Christopher Kruegel. Denver, CO, USA: ACM Press, October 2015, pp. 591–602. DOI: [10.1145/2810103.2813705](https://doi.org/10.1145/2810103.2813705). Also at ia.cr/2015/931.
- [RR21] Mike Rosulek and Lawrence Roy. “Three Halves Make a Whole? Beating the Half-Gates Lower Bound for Garbled Circuits”. In: *Advances in Cryptology – CRYPTO 2021, Part I*. Ed. by Tal Malkin and Chris Peikert. Vol. 12825. Lecture Notes in Computer Science. Virtual Event: Springer, Cham, Switzerland, August 2021, pp. 94–124. DOI: [10.1007/978-3-030-84242-0_5](https://doi.org/10.1007/978-3-030-84242-0_5). Also at ia.cr/2021/749.
- [NIST-IR8214C] Luís T. A. N. Brandão and René Peralta. *NIST First Call for Multi-Party Threshold Schemes*. (National Institute of Standards and Technology) NIST Internal Report (NISTIR) 8214C. 2025. DOI: [10.6028/NIST.IR.8214C](https://doi.org/10.6028/NIST.IR.8214C).