Off-Key: Inconsistent MAC Keys Cause Selective-Failure Attacks in SPDZ-Family MPC

Marcel Keller
CSIRO Technology

Sela Navot
Surfboard AI

Nirvan Tyagi
University of Washington

OUR CONTRIBUTION

SPDZ-family protocols are secure only if every value is authenticated under one shared MAC key. We show that implementations let this key differ between batches of preprocessing, so a corrupted party can make the protocol's abort depend on an honest party's input: a selective-failure attack that breaks SPDZ's privacy guarantee.

What is secure multi-party computation?

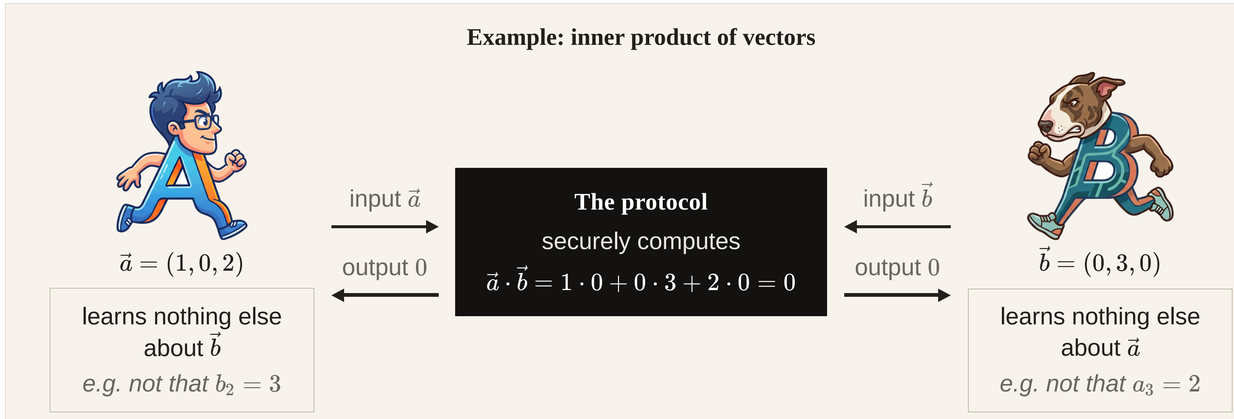
Multi-party computation (MPC) lets parties compute an agreed upon function on their private data, learning the output of the function but **nothing else about the input**. It is increasingly deployed for privacy-preserving analytics over data no single party will reveal, from user telemetry to ad measurement.

A protocol for a function $f(a, b)$ is secure in the malicious setting (MPC with abort) if:

> **Correctness.** If the protocol completes, the final output of the computation is correct.

> **Privacy.** Party learns nothing about the other's input beyond what the agreed output $f(a, b)$ reveals.

In this setting, MPC with abort, an adversary can halt the protocol. However, an adversary **cannot force a selective abort based on the honest parties' inputs**.



SPDZ: efficient MPC for arithmetic circuits

SPDZ writes f as an **arithmetic circuit** (additions and multiplications over a large finite field) and evaluates it directly on **additive secret shares**, one per party, reconstructing only the outputs.

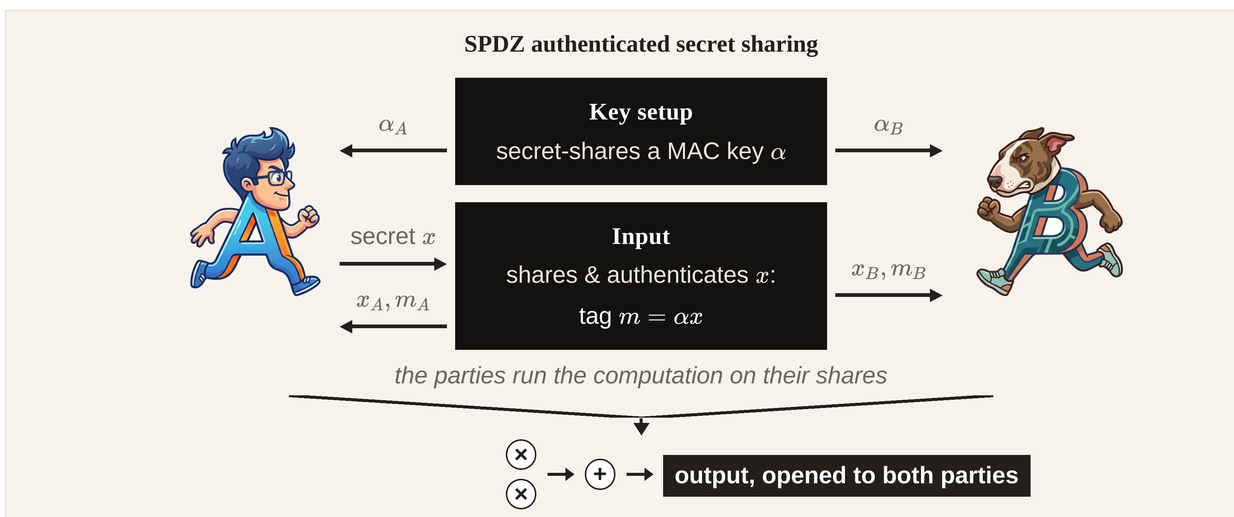
Authentication. Plain shares are malleable: a party could tamper before opening (e.g. add 1 to the output). SPDZ prevents this by tagging every value under a shared **MAC key** α that neither party knows, with a **tag** $m = \alpha x$.

$$x = x_A + x_B \qquad m = \alpha x = m_A + m_B \qquad \alpha = \alpha_A + \alpha_B$$

each party holds only its own shares

Opened values are **checked** against the key; a bad tag **aborts** the protocol.

> A value is **valid** only if its tag matches under the one global key α:
>
> $$m_A + m_B = (\alpha_A + \alpha_B)(x_A + x_B)$$



The attack: when the key is inconsistent

> SPDZ's security assumes that **if a value is tagged with one key α, its validity will only be checked with respect to that key α.**

A cheating party breaks this with two changes: the value is checked under a **different key**, and it adds a guess g to its own tag share.

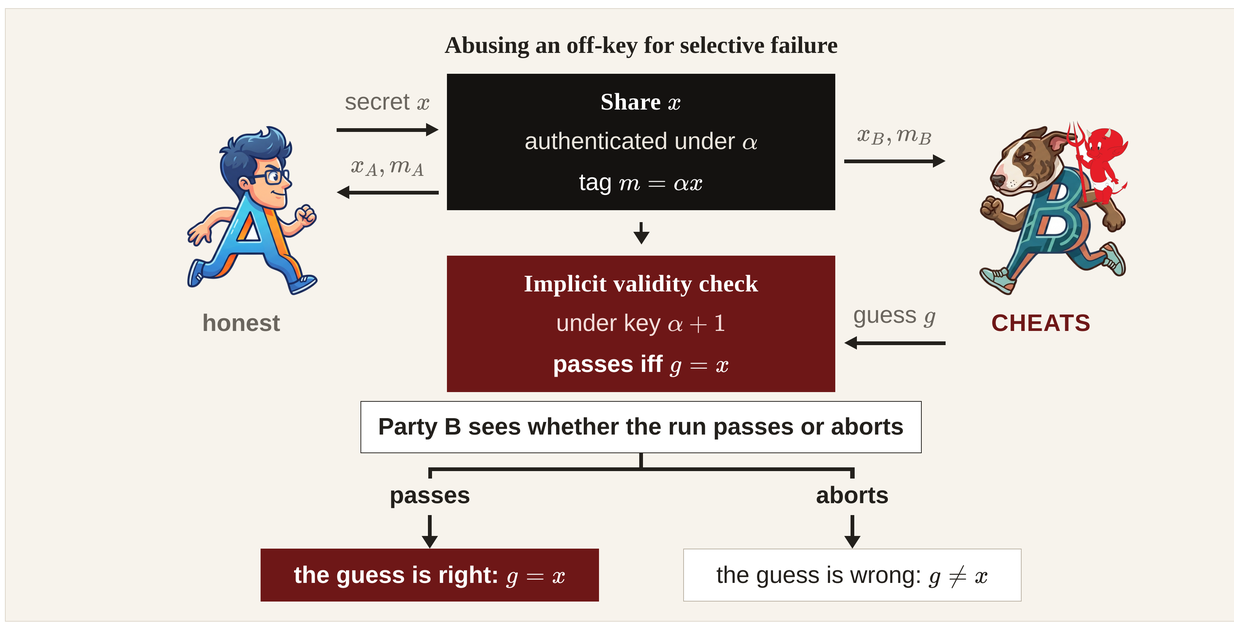
$$\alpha \longrightarrow \alpha + 1 \qquad m \longrightarrow m + g$$

the check key is incremented by 1; the guess g is added to the MAC

The opened tag is then $\alpha x + g$, while the check under $\alpha + 1$ expects $(\alpha + 1)x$:

$$\alpha x + g \;=\; (\alpha + 1)\, x \quad\Longleftrightarrow\quad g = x$$

passes exactly when the guess is right, revealing whether the adversary correctly guessed x.



In practice: attacking MPC frameworks

To make the attack work, a cheating party needs a value that was **authenticated under one key** but is **checked under a different one**: the tag on it uses α, while the check uses $\alpha + 1$.

This requires the framework to generate pre-processing material with respect to two different keys.

SPDZ pre-processing protocols prevent this in theory, but this is not well enforced in practice.

> **The fix.** Implementations must enforce that **the key remains the same across runs**. This generally involves persisting the commitment to the key, as opposed to only the key's plaintext representation.

Framework	How the keys come to differ
MP-SPDZ	each thread and run draws fresh key material; if one holds α, the next can hold $\alpha + 1$ when an adversary increments its share at the right time
SCALE-MAMBA	a large computation can span several runs, each re-reading the key from a plaintext file; an adversary can modify its key share between runs
FANNG-MPC	a fork of SCALE-MAMBA, with the same key handling

Disclosure

We reported the issue to all relevant projects. **MP-SPDZ** (whose maintainer is a co-author) patched it; **SCALE-MAMBA** is unmaintained; and **FANNG-MPC**'s maintainers acknowledged it and cleared us to publish.

This work does not invalidate any security proofs, of SPDZ or other protocols. Rather, it exploits an under-explored invariant that was not maintained in implementations.

Acknowledgements

We thank **Xiangfu Song** for suggesting that inconsistent keys could lead to issues in SPDZ.

Learn more

Proof of concept

One-pager abstract

REFERENCES

SPDZ protocol: Damgård, Pastro, Smart, Zakarias. Multiparty Computation from Somewhat Homomorphic Encryption. CRYPTO 2012.
Damgård, Keller, Larraia, Pastro, Scholl, Smart. Practical Covertly Secure MPC for Dishonest Majority – or: Breaking the SPDZ Limits. ESORICS 2013.

MP-SPDZ library: Keller. MP-SPDZ: A Versatile Framework for Multi-Party Computation. ACM CCS 2020.
SCALE-MAMBA library: Aly et al. SCALE-MAMBA v1.14: Documentation. 2021.
FANNG-MPC library: Aaraj et al. FANNG-MPC: Framework for Artificial Neural Networks and Generic MPC. IACR TCHES 2025.