SAMBA: Enabling Confidential and Collaborative Cloud Functions

PI: Stephen Herwig, Assistant Professor, Department of Computer Science, William & Mary
Cash funding needed: 79,503 USD
AWS promotional credits needed: 10,713 USD

Abstract — We propose SAMBA, a Function-as-a-Service system that ensures data confidentiality and function order within a function chain, even with an untrusted cloud provider or distrusting parties. SAMBA leverages AMD SEV-SNP trusted execution environments to create secure function sandboxes, and uses aggregate signatures for control flow integrity. To enable scaling and replication without a trusted key escrow, Samba employs proxy re-encryption, allowing any function replica to decrypt messages for its target function.

Keywords: Confidential Computing, Cloud Functions, Proxy Re-encryption, Aggregate Signatures

1 Introduction

Function-as-a-Service (FaaS) platforms like AWS Lambda are popular choices for building event-driven, cloud-based solutions because of their autoscaling capabilities and pay-per-use billing. The stateless nature and loose coupling of functions allow organizations to create complex workflows simply by linking multiple functions in a sequence, or *function chain*. A recent study found that 46% of real-world FaaS applications use function chains, with some chains containing up to 10 functions [1]. Despite the name, function chains may be input-dependent, allowing for branching and looping, and are thus more accurately graphs.

There are many use cases—particularly in highly-regulated sectors like government, finance, and healthcare—, where function chains must process privacy-sensitive data, and where the chain includes functions from multiple organizations. For instance, a chain to process a loan application could include: (1) the bank's initial function to process the application, (2) a government service's function to verify the applicant's identity, and (3) a credit bureau's function to retrieve the applicant's credit score. In such a setting, each organization must trust another's function with their data. Even with mutual trust among organizations, adversaries can exploit vulnerabilities in functions, manipulating data flows to steal sensitive data and conduct stealthy operations [2]. Moreover, despite seemingly trustworthy cloud providers, customers must still be wary of cloud infrastructure bugs [3], insider threats [4], and data disclosures to law enforcement [5]–[7].

Prior work in securing function chains focuses either on guaranteeing the chain's data confidentiality, or the integrity of function order. For data confidentiality, previous research systems [8]–[14] leverage hardware trusted execution environments (TEEs, particularly Intel SGX enclaves [15], [16]) to enable confidential function workflows. Unfortunately, to support autoscaling, these systems rely on a trusted key distribution enclave to provision each function replica with the same keying material, creating a key escrow and single point of failure. For function order integrity, research systems like Kalium [17] and Valve [18] model the function chain as a call graph, and apply control flow integrity techniques to ensure that runtime behavior conforms to the graph. Since a function has only a local view of the application, these systems also must rely on a trusted, global controller to track the application-wide control flow.

Our research will guarantee that function chains do not leak sensitive data to unattended parties—including the cloud and function providers—without resorting to centralized trusted services.

Our insight is to combine non-interactive cryptographic protocols with TEEs to secure function chains without centralized trust. We introduce SAMBA, the first FaaS platform to guarantee data confidentiality and path integrity without relying on a trusted control plane. In SAMBA, each function replica runs in a TEE and independently generates its own keys. Using proxy re-encryption, SAMBA enables any replica to decrypt messages intended for its target function, eliminating the need for a trusted key escrow. For control flow integrity, Samba's runtime cryptographically signs each function's provenance, removing the need for a global controller. To minimize bandwidth and validation overhead, Samba uses aggregate signatures [19] to compress the signatures of the entire chain into a single, compact signature.

1.1 Background

Confidential computing. *Confidential computing* protects data in use by processing it within a hardware TEE, isolating it from unauthorized access or modification. This proposal focuses on the latest generation of TEEs—specifically AMD SEV-SNP [20]—which encrypts the memory of an entire guest VM to create *confidential VMs*. A critical feature of AMD SEV-SNP (and all TEEs) is *remote attestation*: the trusted hardware measures (computes a hash of) the VM workload, and a unique hardware key signs this initial state, producing an *attestation report*. Thus, AMD serves as the root of trust, eliminating the need to trust the server owner, while customers can retrieve the attestation report to verify that the correct software is running within the confidential VM. To allow an attestation to bind a runtime value, the confidential VM can include a small amount of *user data* in the signed report.

Proxy re-encryption. *Proxy re-encryption* is a cryptographic scheme that allows an untrusted proxy to convert a ciphertext encrypted under Alice's public key into a ciphertext that Bob can decrypt with his secret key, without learning the underlying plaintext. At a high-level, Alice and Bob construct a public *re-encryption key* $RK_{\text{Alice}\to\text{Bob}}$ that the proxy uses to re-encrypt the ciphertext from Alice (the *delegator*) to Bob (the *delegatee*). In 1998, Blaze, Bleumer, & Strauss [21] designed the first proxy re-encryption construction based on the ElGamal encryption system [22]. Shortly thereafter, Dodis and Ivan [23] developed a unidirectional variant, and later Ateniese et al. [24] applied bilinear maps [25] (and specifically BLS signatures [19]) to develop schemes that did not require interaction between the delegator and delegatee. Since then, numerous works have explored features like multiple re-encryptions [26] and revocation [27], and security properties like chosen-ciphertext resistance [28] and unlinkability of ciphertexts [29].

1.2 Threat Model

Our system has three parties: the *cloud provider*, the *function providers*, and *external attackers*. A function chain may contain functions from multiple providers. For any given function provider, the potential adversaries are the other function providers in the chain, the cloud provider, and external attackers. The goal of an adversary is to learn the inputs or outputs of a (peer) function, modify these inputs or outputs, or modify the sequence of functions in the chain (e.g., to insert a function that sends data to an adversary-controlled log). A function provider can submit any function to the chain, including a malicious function that tries to leak data or subvert the cloud provider. We assume that functions may contain bugs that unintentionally leak data or expose the function to exploitation. We assume an adversary cannot breach the security of confidential VMs; we trust the system software in the VM, and consider side-channel attacks out-of-scope.

2 Methods

Trusted function runtime. For SAMBA's function runtime, we will extend *Project Oak*,[1] an open-source framework for processing private data within a TEE. The central component of Project Oak is the *Oak Functions* platform (see Figure 1), which executes each function in a confidential VM with a custom, minimal operating system kernel designed to execute only a single process. The function runs in a secure WebAssembly (Wasm) runtime sandbox, preventing the process from leaking any sensitive client data. Additionally, Oak uses the DICE [30] architecture for measured boot to extend AMD SEV-SNP's attestation from the initial state of the VM to the entire VM workload. We will modify the Oak kernel with cryptographic support to transparently manage keys and encrypt I/O.

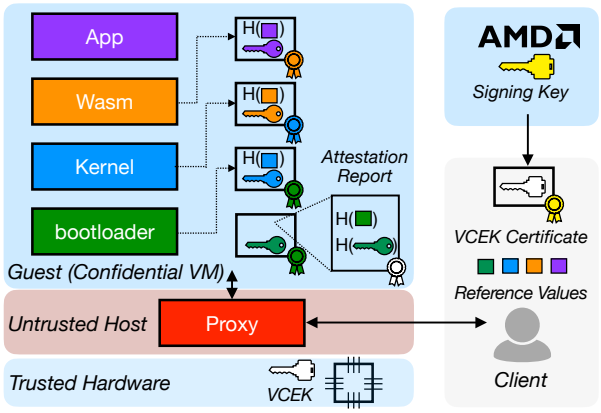

Figure 1: Oak and DICE architecture. In the DICE model, each software layer loads and measures the next layer, generates an ephemeral key pair for the layer, and issues the layer a certificate endorsing its measurement and public key. Oak uses AMD's trusted hardware as the root of trust.

Key management. In previous research, each function replica would attest to a trusted key server to obtain a shared key pair. To eliminate this central, trusted dependency, SAMBA uses decentralized key management with proxy re-encryption. As Figure 2 illustrates, the first instance of a function generates a key pair and registers the public key in an untrusted registry, along with an attestation report that proves the key was generated in an Oak TEE. When a new replica is launched, the replica registers its own public key, prompting the original instance to generate a re-encryption key. As a result, each function can encrypt messages for the original downstream instance's public key, remaining independent of that function's scaling.

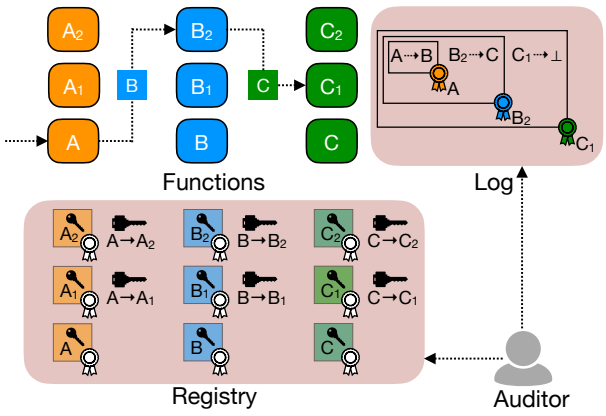

Figure 2: SAMBA architecture.

Control flow integrity. Beyond encrypting function I/O, SAMBA must ensure the integrity of the sequence of functions. SAMBA approaches this challenge in two ways. First, we extend the Oak runtime to include a cryptographically binding record of provenance: each function extends a path signature by signing over its current link. The last function then posts this provenance record to an untrusted log that any organization can monitor and audit. Additionally, if an organization knows a flow graph for the function chain (or some subset thereof), Oak will locally verify that the received event and the function's subsequent output conform to the graph.

Optimization. SAMBA relies on certificate chains both for attestation and provenance. To reduce bandwidth and storage costs, SAMBA can compress the chains using an aggregate signature

[1] `https://github.com/project-oak/oak`

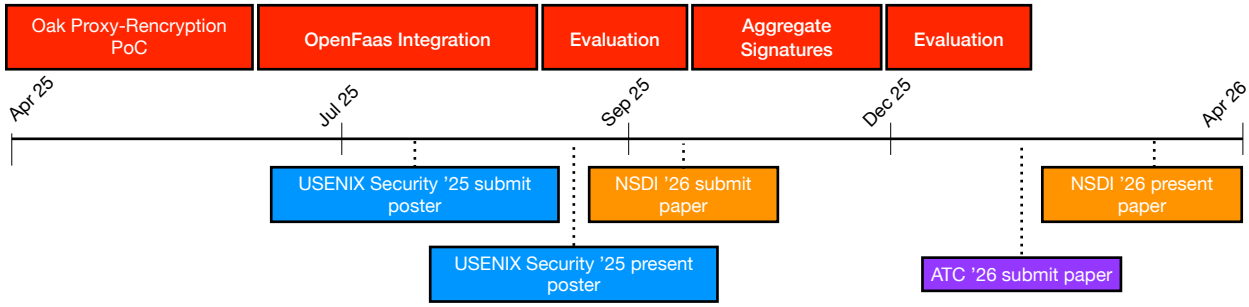

Figure 3: Estimated project timeline and milestones.

scheme [19]. In an *aggregate signature*, each private key sk_i signs a *distinct* message m_i to form signature σ_i, and any party can compress the σ_i into a single, small aggregate signature σ^*. The aggregate signature convinces a verifier that each signer signed their respective message.

3 Expected Results

Implementation. We will implement SAMBA within Oak and the OpenFaaS [2] serverless platform. For proxy re-encryption, we will explore integrating both bilinear map [24] and post-quantum lattice-based schemes [26]. For aggregate signatures, we will use BLS signatures [19], [31].

Evaluation. We will assess SAMBA's performance by measuring its overhead on open-source AWS Lambda serverless applications, such as CodePipeline[3] and MapReduce[4], which differ in chain length and complexity. To decompose the overheads, we will compare SAMBA with: *(1)* SAMBA-strawman, which provides each function replica with the same key pair via an enclaved key service; *(2)* SAMBA-untrusted, which runs functions in a non-confidential Oak VM; and *(3)* standard OpenFaaS, using OpenFaaS's default function runtime. We will utilize the `wrk`[5] and Grafana `k6`[6] benchmarking tools to measure end-to-end latency, cold start times, and scalability.

Timeline. Figure 3 shows an anticipated timeline for the period of performance. The first half of the project develops the initial SAMBA prototype, with a goal of submitting to NSDI '26. We will also submit a poster presentation of this work to USENIX Security '25. The second half modifies the prototype to replace the public log with an aggregatable signature of the function chain sequence, with the intent of submitting to ATC '26.

[2] `https://www.openfaas.com`
[3] `https://github.com/aws-samples/aws-codepipeline-stepfunctions`
[4] `https://github.com/awslabs/lambda-refarch-mapreduce`
[5] `https://github.com/wg/wrk`
[6] `https://grafana.com/oss/k6/`

A References

[1] M. Shahrad, R. Fonseca, I. Goiri, *et al.*, "Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider," in *USENIX Annual Technical Conference (ATC)*, 2020.

[2] M. M. Ahmadpanah, D. Hedin, M. Balliu, L. E. Olsson, and A. Sabelfeld, "SandTrap: Securing JavaScript-driven trigger-action platforms," in *USENIX Security Symposium*, 2021.

[3] H. S. Gunawi, M. Hao, T. Leesatapornwongsa, *et al.*, "What bugs live in the cloud? A study of 3000+ issues in cloud systems," in *ACM Symposium on Cloud Computing (SOCC)*, 2014.

[4] N. Santos, K. P. Gummadi, and R. Rodrigues, "Towards trusted cloud computing," in *USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2009.

[5] *Amazon gave Ring videos to police without owners' permission*, `https://www.politico.com/news/2022/07/13/amazon-gave-ring-videos-to-police-without-owners-permission-00045513`, 2022.

[6] *US court forces Microsoft to hand over personal data from Irish server*, `https://www.theguardian.com/technology/2014/apr/29/us-court-microsoft-personal-data-emails-irish-server`, 2014.

[7] *The police want your phone data. Here's what they can get — and what they can't.* `https://www.vox.com/recode/2020/2/24/21133600/police-fbi-phone-search-protests-password-rights`, 2020.

[8] T. Hunt, Z. Zhu, Y. Xu, S. Peter, and E. Witchel, "Ryoan: A distributed sandbox for untrusted computation on secret data," in *Symposium on Operating Systems Design and Implementation (OSDI)*, 2016.

[9] F. Alder, N. Asokan, A. Kurnikov, A. Paverd, and M. Steiner, "S-FaaS: Trustworthy and accountable function-as-a-service using Intel SGX," in *ACM Workshop on Cloud Computing Security (CCSW)*, 2019.

[10] B. Trach, O. Oleksenko, F. Gregor, P. Bhatotia, and C. Fetzer, "Clemmys: Towards secure remote execution in FaaS," in *ACM International Systems and Storage Conference (SYSTOR)*, 2019.

[11] S. Brenner and R. Kapitza, "Trust more, serverless," in *ACM International Systems and Storage Conference (SYSTOR)*, 2019.

[12] M. Li, Y. Xia, and H. Chen, "Confidential serverless made efficient with plug-in enclaves," in *International Symposium on Computer Architecture (ISCA)*, 2021.

[13] S. Zhao, P. Xu, G. Chen, M. Zhang, Y. Zhang, and Z. Lin, "Reusable enclaves for confidential serverless computing," in *USENIX Security Symposium*, 2023.

[14] S.-J. Kim, M. You, B. J. Kim, and S. Shin, "Cryonics: Trustworthy function-as-a-service using snapshot-based enclaves," in *ACM Symposium on Cloud Computing (SOCC)*, 2023.

[15] F. McKeen, I. Alexandrovich, A. Berenzon, *et al.*, "Innovative instructions and software model for isolated execution," in *Workshop on Hardware and Architectural Support for Security and Privacy (HASP)*, 2013.

[16] V. Costan and S. Devadas, "Intel SGX Explained," Cryptology ePrint Archive, Tech. Rep. 2016/086, 2016.

[17] D. S. Jegan, L. Wang, S. Bhagat, and M. Swift, "Guarding serverless applications with Kalium," in *USENIX Security Symposium*, 2023.

[18] P. Datta, P. Kumar, T. Morris, M. Grace, A. Rahmati, and A. Bates, "Valve: Securing function workflows on serverless computing platforms," in *The Web Conference (WWW)*, 2020.

[19] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "Aggregate and verifiably encrypted signatures from bilinear maps," in *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2003.

[20] AMD, "AMD SEV-SNP: Strengthening VM isolation with integrity protection and more," AMD, Tech. Rep., 2020.

[21] M. Blaze, G. Bleumer, and M. Strauss, "Divertible protocols and atomic proxy cryptography," in *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 1998.

[22] T. Elgamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, vol. 31, no. 4, Jul. 1985.

[23] A.-A. Ivan and Y. Dodis, "Proxy cryptography revisited," in *Network and Distributed System Security Symposium (NDSS)*, 2003.

[24] G. Ateniese, K. Fu, M. Green, and S. Hohenberger, "Improved proxy re-encryption schemes with applications to secure distributed storage," in *Network and Distributed System Security Symposium (NDSS)*, 2005.

[25] D. Boneh and M. K. Franklin, "Identity-based encryption from the Weil pairing," in *International Cryptology Conference (CRYPTO)*, 2001.

[26] Y. Polyakov, K. Rohloff, G. Sahu, and V. Vaikuntanathan, "Fast proxy re-encryption for publish/subscribe systems," *ACM Transactions on Privacy and Security*, vol. 20, no. 4, 2017.

[27] A. Sahai, H. Seyalioglu, and B. Waters, "Dynamic credentials and ciphertext delegation for attribute-based encryption," in *International Cryptology Conference (CRYPTO)*, 2012.

[28] R. Canetti and S. Hohenberger, "Chosen-ciphertext secure proxy re-encryption," in *ACM Conference on Computer and Communications Security (CCS)*, 2007.

[29] A. Davidson, A. Deo, E. Lee, and K. Martin, "Strong post-compromise secure proxy re-encryption," in *Australasian Conference on Information Security and Privacy (ACISP)*, 2019.

[30] *DICE attestation architecture*, `https://trustedcomputinggroup.org/resource/dice-attestation-architecture/`, Version 1.1, Jul. 2024.

[31] A. Boldyreva, "Threshold signatures, multisignatures and blind signatures based on the Gap-Diffie-Hellman-Group signature scheme," in *International Workshop on Public Key Cryptography (PKC)*, 2003.