Decentralized Certificate Authorities (Poster Proposal)

Hannah Li, Bargav Jayaraman, David Evans
University of Virginia
https://oblivc.org/dca

The security of TLS depends on trust in certificate authorities, and that trust stems from their ability to protect and control the use of a private signing key. The compromise of a CA private key represents a single point-of-failure that could have disastrous consequences, so CAs go to great lengths to attempt to protect and control the use of their private keys. Nevertheless, keys are sometimes exposed and may be misused accidentally or intentionally by insiders.

We propose a new model where a CA's private key is split among multiple parties, and signatures are produced using a secure multi-party computation protocol that never exposes the actual signing key. Decentralized CA can be used in three different settings: jointly computing the signature and public key by (1) a single CA that wants to protect its own signing keys by distributing them among its own hosts, (2) two independent CAs who want to jointly mitigate the risks of key exposure or misuse, and (3) a CA and subject who wants to protect itself from a CA generating rogue certificates by being directly involved in the certificate signing process. We have build a prototype implementation to demonstrate the practicality of this concept using secure two-party computation to generate certificates signed using ECDSA on curve `secp192k1`.

Distributed Key Generation. To generate a joint public key for signature verification, the two hosts operating as a joint CA independently generate random private key shares sk_A and sk_B which are combined in a multi-party computation to obtain master private key sk. This private key is never revealed outside the MPC. The corresponding public key is obtained via curve point multiplication as $pk = sk \times G$ within the MPC, and revealed to both hosts who publish it as a joint public key.

Distributed Certificate Signing. Each host generates a hash of the to-be-signed certificate, and inputs that value along with its private key share and one-time signing key share into an MPC protocol (Figure 1). An equality test ($z_A == z_B$), is performed inside the MPC to ensure both the CAs agree upon the certificate to be signed. The key shares are combined in the MPC, and used for a standard ECDSA algorithm to generate the signature which is revealed at the end of the protocol.

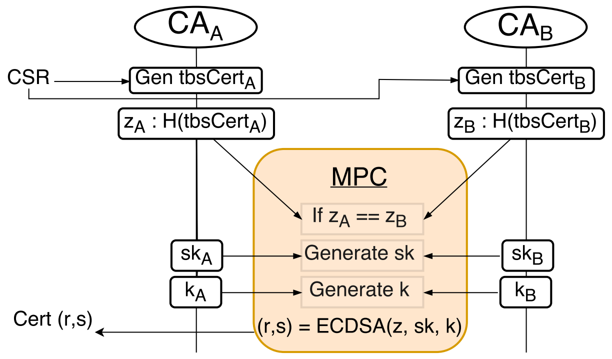

Figure 1: Distributed Certificate Signing

Implementation. We use the Obliv-C framework that includes the latest enhancements to Yao's garbled circuits protocol. Apart from the Yao's protocol for garbled circuits that is secure against semi-honest adversaries, Obliv-C also implements the dual execution protocol to provide security against active adversaries.

Cost. The most expensive step in ECDSA (for both key generation and signing) is the curve point multiplication protocol, which accounts for 19.2 billion of the 22 billion total garbled gates needed to execute the signing protocol. The cost is dominated by the bandwidth required to send the signing circuit, which requires 410 GiB to transmit using 80-bit wire labels and the half-gates method.

We have conducted experiments using Amazon AWS EC2 nodes to jointly sign a certificate with different scenarios to model two hosts in the same data center (perhaps reasonable for a CA protecting its own signing key), two hosts in different regions, and hosts using different cloud providers. Using the semi-honest Yao's protocol for multi-party computation, the signing costs around 30 cents per certificate when bandwidth is free (both hosts in the same data center), up to around $40 per certificate for external market-rate bandwidth. These costs approximately double for signing with dual execution protocol, but still well within what we believe is practical for high-value certificates.