Lane Gramling
EECS 700
Reading 18

Summary - The Pythia PRF Service

This paper tackles the issue of hardening sensitive data from brute-force attacks typically carried out offline on stolen databases. The core philosophy of the Pythia service is to encrypt messages in such a manner that requires information only available through a server hosting the Pythia API in order to successfully decrypt data that stored in a database. The paper proposes a new cryptographic primitive for pseudorandom functions called a verifiable partially-oblivious PRF.

Contributions

As hinted in the summary, the key contribution of the paper is the use of verifiable partially-oblivious PRFs in encryption & decryption of protected data. This requires information only accessible from the Pythia API, which enables the service to act as a secured third party capable of monitoring for attempts to brute-force data that may have hypothetically been stolen from a database using it. Secure operation of this scheme is carried out by revealing a portion of the input, referred to as the *tweak*, to the service as a requirement in encrypting or decrypting messages. The tweak is paired with the message in question, while the message itself remains unknown to the user. This makes space for fine-grained rate limiting to be implemented into the API, which enables the monitoring and throttling of queries exhibiting anomalous volumes or other unusual patterns of activity. Additionally, this structure allows for key updates to be performed quickly and efficiently without requiring the entire set of users to change their passwords (or whatever type of sensitive data is being stored). The authors also contribute a rigorous description of the mathematical foundations proving the security of this technique.

Summarizing the authors' outline on security details: the message is blinded from the server when the client raises the message to a randomly selected exponent before communicating to the server, which is a bijective function allowing unblinding by the client at a later time; additionally, a zero-knowledge proof of correctness, of which I concede evades my scope of understanding, is then computed by the server on the output of the encryption/decryption computation performed on the message, verifying the function's correctness before communicating back to the client. A couple more variants which do not rely on blinding techniques are also provided by the authors, as well as a demonstration of how Pythia can be used in concert with password onion techniques.

Beyond these contributions, the authors also provide an elaboration on the implementation of their proposed service including example use cases; likewise, metrics are provided showing respectable performance achieved, with microsecond-order hashing times and millisecond-order mean round-trip latencies when implemented on a standard AWS EC2 instance hosted in California and accessed from Madison, which certainly passes as a reasonable average access scenario - if not even further from best-case than an optimized implementation used in practice by a hypothetical company or service implementing this system. On top of the previously mentioned benefits to enforcing rate-limiting in the API, it also allows for more stable performance accompanying its added security.

Limitations

As addressed by the authors, the use of rate-limiting in the API (and a third party in general) exposes a vulnerability to targeted denial-of-service attacks, although this does not compromise integrity of the data and can be hedged against with added defenses. Additionally, the introduction of a third party allows for a single point of failure - if the service for some reason became unavailable, this would severely inhibit if not cut off access to services relying on it in something like their login process.

Future Work

Most of the potential for future work on this idea, in my opinion, lies in expanding the implementation for better decentralizing its functionality and other methods of decreasing its risk as a single point of failure. Unless I am overlooking something, this should be possible with standard load distribution techniques, and does not exhibit any traits that stand out as major obstacles in decentralization

(leaving in the possibility for little tweaks and extra implementation details to allow for distributed storage and access). Beyond this, as the mathematical rigor of the offered security benefits on the computational layer was demonstrated, it feels likely that future work with the best work-to-reward ratio is in investigating weaknesses on the physical layer of the implementation, and security risks of an added third party, regardless of whether communications between the client and third party are sufficiently encrypted.