CryptID - Distributed Identity Management Infrastructure

Jan-Ole Malchow
Secure Identity Research Group
Freie Universität Berlin
jan-ole.malchow@fu-berlin.de

Volker Roth
Secure Identity Research Group
Freie Universität Berlin
volker.roth@fu-berlin.de

***Abstract*—Many of the services on which we depend on the Internet where designed when communications security was not a major concern. The toll for retrofitted security was increased complexity. When search engines emerged users began to type only significant parts of a domain name into the search field and clicked on the appropriate link. In this poster we argue that this paradigm shift ultimately allows us to disentangle, replace and simplify the existing stack of Internet services related to name services and security.**

I. Introduction

Many of the services on which we depend on the Internet where designed at a time when communications security was not as much a concern as it is today. As security became increasingly important, existing services where augmented and retrofitted with security features and protocols. The toll was increased complexity and increasing difficulty to manage and maintain the resulting protocol stack. Specific examples are the design of DNSSEC based on the original DNS augmented with PKI technology from the X.509 standard (PKIX). Another example is the proliferation of SSL/TLS as the go-to technology to establish encrypted communication, which relies on PKIX as well. Unfortunately, PKIX is dominated by a few large players and the practice of PKIX has become the target on significant criticism over technological, organizational and market failures [1], [2]. Prior art in the field, as summarized in [3], focused on fixing these problems by means of additional components and as a consequence more complexity. Meanwhile, the need for DNS is eroding. Once it was conceived as a means to use human-readable names for routing adresses such as IP numbers. When the web emerged, users had to type fully qualified domain names into URL bars in order to navigate to a website. When search engines emerged, user behavior changed. Users began to type only significant parts of a domain name into the search field and clicked on the appropriate link in the search results in order to navigate to the intended site. This eventually led to the fusion of search fields and URL bars into a single field, for example, Chrome's *Omnibox.* The latest step in the evolution of omniboxes is to shun domain names in favor of displaying components of *distinguished names* from Extended Validation certificates. In this poster we argue that this paradigm shift ultimately allows us to 1) replace human-readable but insecure names with secure but random-looking identifiers, and to 2) disentangle, replace and simplify the existing stack of Internet services related to name services and security. We refer to our proposed replacement as *CryptID.* In CryptID, each entity (client or server) has a unique and provable ID, that is, a public key. Furthermore, each entity has a routing address that allows messages to flow from one entity to another, for example, an IP address or even a postal address. By default, we assume that entities wish to connect to each other in a fashion that is confidential, integer and authentic. In order to do so, entities need the routing addresses. Once established, the connection is secured easily using the server's unique ID. But how does an entity find an ID and how does the entity know the ID belongs to the intended partner? CryptID deals with these two questions separately. Finding IDs is solved by means of indexes and dictionaries. While trust establishment is solved in a demand aware fashion.

Finding IDs – An index provides service entries that map descriptions to IDs. By signing an entry with its ID, a service can prevent forgery of its entry. Descriptions are limited in length but their structure is not mandated. By signing an entry with its ID, a service can prevent forgery of its entry. The index provides indexing and searching facilities. Additionally, an index vets its entries by connecting to the service and by verifying that the (authenticated) connection matches the claimed ID. In order to connect to a service, the index makes use of dictionaries. A dictionary provides routing entries that map IDs to routing addresses. Dictionaries can be implemented in a decentralized and highly scalable fashion. Service entries and routing entries are self-authenticating, that is, signed. This allows entities to update both types of entries in a fashion that does not require indexes and dictionaries to trust individual entities. Clients search for IDs in indexes and they cache the found IDs locally using names with a local scope (*pet names*), that is, two clients may use different names for the same ID but each name is unique within its local scope.

Trusting IDs – Indexes and dictionaries do not solve the trust issue. They yield a clean architecture and highly scalable infrastructure that separates trust concerns from infrastructure operations. Anyone can operate indexes or dictionaries. Indexes and dictionaries can be deployed in a fashion that is redundant or is location-based. Our goal with CryptID is to explore the design space that emerges from this simple architecture. For example, vetting services may evolve in a sector-specific fashion and they may leverage existing trust relationships among organizations. Consider a banking association that vets the entries of its member banks, and member banks may vet the entry of its association. Hence, knowing one bank means one knows them all, figuratively speaking. Prior art has looked at

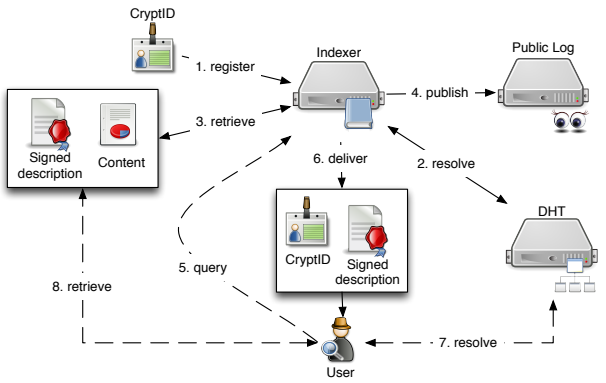

Fig. 1. Registration and retrieval of services. Solid lines show the registration process. Dashed lines are part of the retrieval process.

trust relationships such as this primarily from a technology perspective. CryptID instead seeks to incentivize independent and alternative vetting services by disentangling infrastructure services from vetting.

II. System Overview

The CryptID system consists of a management component, a service registration and indexing component, and a routing address resolution component. Users interact with CryptID by means of the identity management. It allows users to: 1) Create and publish CryptIDs, 2) Manage indexers (use or blacklist), 3) Register services at indexers, 4) Manage pet names, and 5) perform searches for services.

The service registration and indexing component is illustrated in Fig. 1. In order to register a new service, the owner submits a valid CryptID to an indexer of his choice (1). A CryptID is valid if an authenticated connection can be established to the resolved routing address (2, 3). A service description for the corresponding CryptID must be part of the service response. Thereby the indexer can authenticate the received description (3). The indexer uses this description as a summary text in search results. The search index uses standard full text indexing. Indexers can form search pools for better results. All data that an indexer returns is signed with its key (6). This allows users to filter out results from undesired indexers that may be in a pool. The indexer publishes a log for each indexed CryptID (4). Each log entry is signed and contains timestamp, nonce (signed by CryptID owner), and the sequence number of the used routing entry.

In order to find a CryptID, a user chooses index servers and submits his full-text query (5). For each search result, the indexers return the signed description and CryptID (6). The results are merged and sorted locally. The user resolves the retrieved CryptID (7) and connects to the desired server (8). CryptIDs are themselves keys in a distributed hash table (DHT). A record can be found for each existing CryptID. If only a public key is available the corresponding CryptID can be easily derived. As such CryptIDs and public keys both resolve to routing addresses and allow secure communication. A CryptID record consists of four main elements: public key, signature, routing addresses and until date.

III. Security Considerations

Registration – The registration process implements public-key challenge-response based authentication to prevent registration of CryptIDs by others than the owner. Only descriptions published by the CryptID owner are accepted by the indexers. This is ensured by verifying the signature of each description. The indexer uses routing addresses from the CryptID to prevent missmatches between description and page content. Ideally the CryptID contains a key for authenticated and encrypted communication between indexer and service. *Monitoring* – Clients require a CryptID to provide a backlink to an indexer log. This forces an attacker to register the CryptID with an indexer. For impersonification an attacker must use a convincing description. This in turn allows the real identity owner to search for entries similar to his description. *Search* – An indexer can not present unintended teaser texts in search results. The user is always able to validate the link between teaser text and CryptID by means of signatures. *Resolving* – The authenticity of each CryptID can be verified based on the signature. Re-usage of old entries is prevented by the until date. Lookups as well as internode communication in the DHT is encrypted. Therefore passive eavesdropping of lookups is prevented.

IV. Current Status

We implemented an indexer prototype based on the Apache Blur stack [5]. This stack offers full-text search on top of a Hadoop cluster. Registration and search of services is already fully implemented. Next steps will we to implement public logging and search pools. A prototype of the identity manager exists as a browser tool. It supports creating, registering and searching of CryptIDs and services. The next steps will be to implement indexer management and pet names. We have an up and running (DHT) implementation written in Go. We designed our DHT based on Kademlia [6]. We extended the default protocol with a cryptographic layer for inter-node communication. The layer does not require additional messages and adds only minimal additional state. For backwards-compatibility and ease of integration, we implemented a DNS proxy that redirects request for the ".cryptid" top-level domain to a DHT node. Therefore CryptID can replace DNS without changes to existing software.

References

[1] Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman, "Analysis of the https certificate ecosystem," in *Proceedings of the 2013 conference on Internet measurement conference*. ACM, 2013, pp. 291–304.

[2] A. Arnbak, H. Asghari, M. Van Eeten, and N. Van Eijk, "Security collapse in the https market," *Communications of the ACM*, vol. 57, no. 10, pp. 47–55, 2014.

[3] T. H.-J. Kim, L.-S. Huang, A. Perring, C. Jackson, and V. Gligor, "Accountable key infrastructure (aki): A proposal for a public-key validation infrastructure," in *Proceedings of the 22nd international conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2013, pp. 679–690.

[4] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*. CRC press, 1996.

[5] The Apache Software Foundation. (2011) Apache blur (incubating) home. [Online]. Available: https://incubator.apache.org/blur/

[6] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric," in *Peer-to-Peer Systems*. Springer, 2002.