# Building a Threshold Cryptographic Distributed HSM with Docker Containers

Caterina Munoz
INRIA Chile,
Avenida Apoquindo 2827, piso 12,
Las Condes, Santiago de Chile.
Email: caterina.munoz@inria.cl

Francisco Montoto
NIC Labs, Universidad de Chile
Blanco Encalada 1975,
Santiago de Chile.
Email: montoto@niclabs.cl

Francisco Cifuentes
NIC Labs, Universidad de Chile
Blanco Encalada 1975,
Santiago de Chile.
Email: francisco@niclabs.cl

Javier Bustos-Jiménez
NIC Labs, Universidad de Chile
Blanco Encalada 1975,
Santiago de Chile.
Email: jbustos@niclabs.cl

***Abstract*—The Domain Name System (DNS) has evolved to support the exponential growth of the Internet, by relying heavily on a highly distributed infrastructure. Nevertheless, trust between server must exist in order to guarantee correct functioning of the system, which is prone to attacks and errors. The Domain Name System Security Extensions (DNSSEC) is the current extension of the DNS system to provide security constrains to the query process. DNSSEC key management main impact on DNS operation has been the use of a monolithic equipment: Hardware Security Modules.**

**A Hardware Security Module (HSM) is a specialized hardware designed to protect keys against logical and physical tampering or extraction, while providing secure mechanisms to employ those keys in cryptographic operations without ever exposing sensitive material. Unfortunately, the high costs of most HSMs make them a reasonable solution only for large corporations. Even then, there is the risk of failures; provisions must then be taken to replace or recover failed HSMs, further increasing the overall cost of this technology.**

**We have presented a distributed signer system based on threshold cryptography, called Poor Man's Hardware Security Module (pmHSM), which provides the signature components of an *HSM* over inexpensive commodity hardware to support the operational signing workflow of DNSSEC. We did test our virtual pmHSM by using it to support the operational signing workflow of DNSSEC. Nevertheless, our solution did not used all the capabilities of the PKCS11 API and it had a single point of failure.**

**Thus, we changed pmHSM's architecture moving part of it services to the client side and isolating the signer, replacing the previous compile-creation version of the distributed signers for self-contained and easy to configure containers. With this change, we aim to build a system more extensible, usable, and more configurable to the users needs.**

978-1-5386-3123-2/17/$31.00 ©2017 IEEE

## I. Introduction

DNSSEC is a security extension to provide authenticity and integrity to the traffic between DNS servers. In regular DNS query/response schema, the answers are not authenticated, making man in the middle attacks a real possibility. In DNSSEC, a public-key infrastructure architecture is introduced to endow every query with an authenticated response, by means of a digital signature and a public key. All these signatures use to be stored safely in Hardware Security Modules (HSM), which often their security level is proportional to their costs.

On one hand an HSM is still a monolithic machine that could fail, and in other hand to put the keys of an inherent distributed protocol as DNS in a monolithic infrastructure seems to be unnatural. Also, there were some issues detected in the use of HSMs. For instance, in the RIPE 62 meeting, Brett Carr declared that "*in September 2010 a HSM hardware failure caused a OS crash, then the HSM was locked on reboot, and the 2-Day TTL on DNSKEY caused slow recovery of .uk signing*" [1]. Also, in ICANN 41 meeting, Vincent Levigneron (from AFNIC) stated

that their "*first DNSSEC outage was in November 2010, during key deletion we had a network issue making our HSM unreachables. The error was not well detected, so the publication process didn't stop as expected. OpenDNSSEC to Bind synchronization process (homemade script) decided to purge the key files one hour after it was supposedly deleted*". Their second outage in February 2011 was caused by a bug in bind, and their third one was in March 2011: "*Bind "decided" to modify it's private records. But at the same time, we had (again) a HSM reachability issue. Then, the published zone was not correct*" [2]. Furthermore, the chance of a HSM failure is considered in the process of signing a root zone in the draft published in 2010 by ICANN[3].

In a previous work [4] we described the *Poor Man's Hardware Security Module* (pmHSM) a system able to emulate a HSM without requiring expensive hardware. We built it by implementing the inner works of the HSM using threshold cryptography with a set of distributed nodes (Figure 1). In order to provide the needed functionality, we distribute the key and the required operations (eg. digital signing) over these nodes. In particular, our system provided digital signing capabilities via an efficient, modular, and diverse implementation of the Victor Shoup's scheme [5].

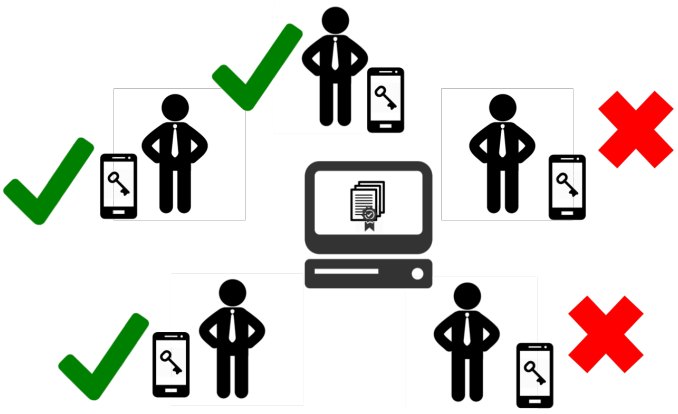


Fig. 1. Threshold Cryptography: Key is divided in $n$ *key-shares* and it is only need $k > \frac{n}{2}$ sign-shares to produce a valid signing.

However, our previous version still presented a single-point of failure (see details in Section IV), so we re-designed it under the following principles:

1) it should be fully distributed,
2) it should be open source, and
3) in order to be adopted by TLD administrators, it deployment should be harmless.

Our solution was designed to be extremely easy to adopt, as it provides a simple yet standard interface for any application that wants to use it: from the outside, the system presents the standard PKCS#11 interface implementing the associated API. This provides a transparent interface between applications and our system: any application which already access an HSM is now be able to use our implementation with almost no changes.

## II. Related Work

Threshold cryptographic systems have been widely used in network protocols. Given its distributed nature it has been widely used in security of wireless [6], [7], [8], and mobile ad-hoc networks [9], [10], [11], voting systems [12], and as a solution for the problem of storing DNS zone secrets online without leaking them to a corrupted server [13].

We remark that our distributed solution can be implemented with no special hardware (which yields smaller costs), is resilient to attacks such as those presented in the work of Bau and Mitchell [14], and provides a high availability rate (such as wanted in the work of Deccio et al. [15]), since nodes can fail and the system will operate anyway [16]. Hardware security modules provide very high mean time between failures, but in the worst case scenario, they can fail or we can suffer natural disasters.

On the DNSSEC side, the two most used mechanisms: Bind [17] and OpenDNSSEC[1] use a one-to-one client-server scheme to manage signatures and keys [18], both provide connection to hardware security modules (HSMs) via PKCS#11 connections. However, the key point in pmHSM is the reliability of the threshold cryptographic backend compared to client-server schemes [19].

## III. Distributed HSM

Our first version of pmHSM [4] comprised of three different process which can run over different machines that communicates using ZeroMQ messages over TCP. The processes are (a) the application, (b) the Manager, and (c), the Node. For each pmHSM instance, there is a single Manager instance and a single set of distributed

[1]http://www.opendnssec.org

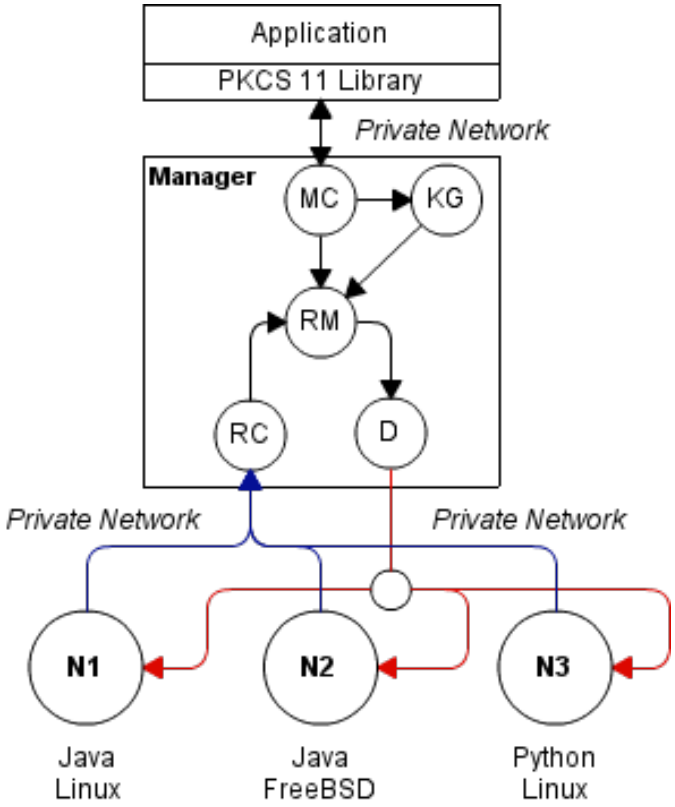



Fig. 2. Architecture of the pmHSM DNSSEC distributed signer

nodes (Figure 2). Notice that multiple applications can send requests to pmHSM.

### A. pmHSM Layers

Our pmHSM was based in a separated implementation of a three layer system [4]:

1) A PKCS11 dynamic library: This is the part that implements the PKCS#11 API. The library file (`libpkcs11.so`) must be installed on the machine with the application that will be making requests to the HSM.
2) A server that acts as a manager of all cryptographic operations.
3) A manager's client application, named *Shareholder node*.

The signing process works as follows: when a node receives a document and a key-share alias, it generated a new signature-share and sends it to the manager. The manager joins the needed signature-shares to make a complete RSA signature. The nodes, as modules that store private key pieces, are the most vulnerable module of the system. Hereby, the system was designed to let the nodes be implemented in different programming languages, and to let them run in different operating systems and architectures.

### B. Inside the Manager

The manager module was developed in Java, and as we stated in [4] it had five sub-modules: Method Collector, Key Generator, Dispatcher, Results Collector y Request Manager. The **Method Collector** was the module that receives the application's requests, and send the response back after the other modules have worked on it using the *Response* mechanism of ZeroMQ. **Dispatcher** was the module that send messages to the nodes using the *Publish* ZeroMQ mechanism. **Results Collector** was the module that listens messages from the nodes in order to execute the requested actions. Finally, **Request Manager** is the module that send specific request to the nodes by the mean of the dispatcher, and generating tickets to be used in the asynchronous processing of the answers.

## IV. System Redesign

We noticed that the manager was a critical part of the system, and it could became a single point of failure of our architecture (Figure 3(a)). Thus, we studied how to improve our system. The first choice was to replicate the manager and use replicate messaging for consistence (Figure 3(b)), and the second choice was the use a replicated database in the manager (Figure 3(c)). Nevertheless both solutions impacted in performance and not solve the problem of the critical part of the infrastructure.

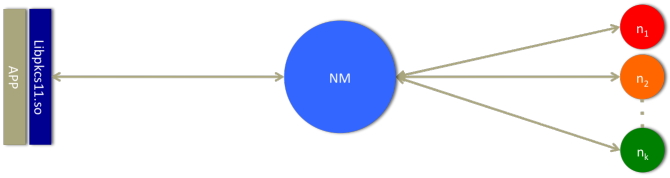
Then, we decided to split the manager services, implementing the dispatcher, key generator, and results collector in the client side with the PKCS#11 API, and distributing main services of method collector and request managers among the client and nodes side. The resultant design is presented in Figure 4 and now the system has not a single point of fairlure.

With the new node's design, we noticed that one of the main drawback in to publish the node "as it is" in our GitHub site [2] was the cryptography library dependencies, which add extra effort to operations and sysadmins in order to install them. Thus, giving our experience with Linux Containers [20], we decided to build Docker containers for the infrastructure, one for the library itself, and one for the signer node.

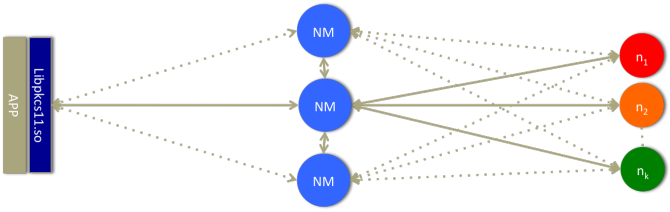
## V. "Containerizing"

We tested the whole system in a desktop computer, with Intel Core i5-2400 CPU at 3.10GHz (x 4), having a docker container for each participant (Knot NameServer

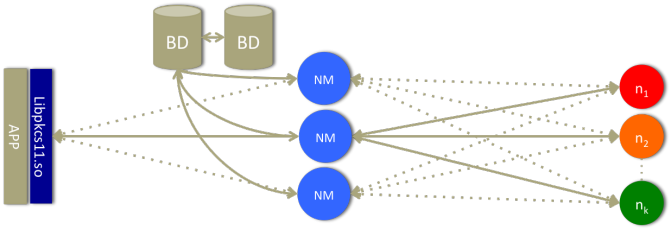
[2]https://github.com/niclabs/tchsm-libdtc




(a) Critic (single) point of failure




(b) Replicated Communication




(c) Replicated Database

Fig. 3. Design process.

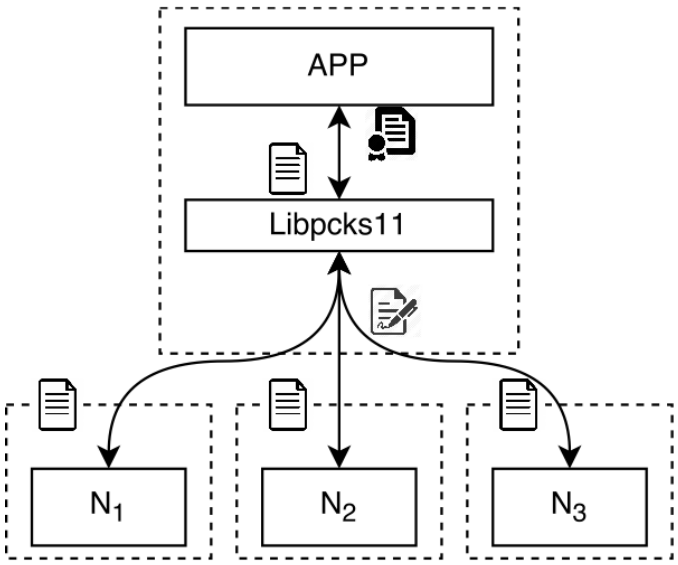



Fig. 4. Architecture of the new pmHSM DNSSEC distributed signer

and four nodes). To avoid nodes to connect to default network, we simply add every component to the same bridge (in our case called `tchsm`):

```
docker network create -d bridge tchsm
for i in $(seq 1 $NODES)
    do
        docker -D run --net=tchsm -d \
        --name node-$i tchsm-node \
        -c /etc/node$i.conf
    done

docker create --net=tchsm \
    --name knot-demo \
    -p ${EXPOSE_PORT}:53 \
    -p ${EXPOSE_PORT}:53/udp \
    tchsm-demo
```

Then, our performance results were:

- Signature: around 75 signatures per second.
- Signing 250,000 domains (500,000 lines in zone file) took 2 hours and 50 minutes.

The simple network configuration provided by Docker and the new self-contained node lead us to improve it and make the node capable now to serve different clients, selecting the *key-share* depending on a `session-id`. With this improvement we can have nodes serving multiple clients, as in Figure 5.

The full system can be tested from https://github.com/niclabs/docker/tree/master/tchsm .

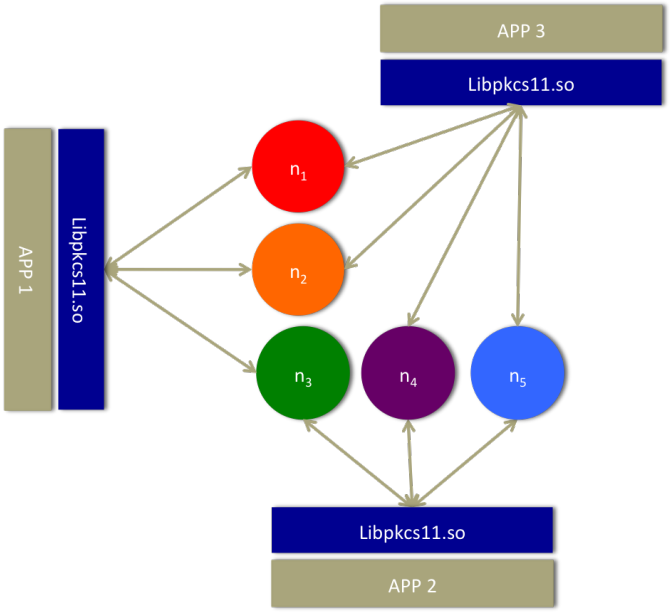



Fig. 5. Multiple client's nodes.

## VI. Conclusions and Future Work

Docker allowed us to make deployments simpler, without requiring a deep intervention of the servers. This is very important when we look for the system to be used by third parties. The migration to Docker does not

cause a decrease in service performance, because the cryptographic operations are, at least, ten times more expensive than the synchronization and communication of the participants.

## REFERENCES


[1] B. Carr, ".uk dnssec status update," in *RIPE 62 Meeting*, 2011, http://ripe62.ripe.net/presentations/219-uk_DNSSEC_Status_Ripe_62.pdf.

[2] V. Levigneron, "Key deletion issues and other dnssec stories," in *ICANN 41 Meeting*, 2011, http://singapore41.icann.org/meetings/singapore2011/presentation-key-deletion-issues-22jun11-en.pdf.

[3] R. D. D. Team, "Dnssec test plan for the root zone (draft)," ICANN and VeriSign, Tech. Rep., 2010, http://www.root-dnssec.org/wp-content/uploads/2010/06/draft-icann-dnssec-testing-01.txt.

[4] F. Cifuentes, A. Hevia, F. Montoto, T. Barros, V. Ramiro, and J. Bustos-Jiménez, "Poor man's hardware security module (pmhsm): A threshold cryptographic backend for dnssec," in *Proceedings of the 9th Latin America Networking Conference*. ACM, 2016, pp. 59–64.

[5] V. Shoup, "Practical threshold signatures," in *Advances in Cryptology—EUROCRYPT 2000*. Springer, 2000, pp. 207–220.

[6] C. Crépeau and C. R. Davis, "A certificate revocation scheme for wireless ad hoc networks," in *Proceedings of the 1st ACM workshop on Security of ad hoc and sensor networks*. ACM, 2003, pp. 54–61.

[7] C. Basile, Z. Kalbarczyk, and R. K. Iyer, "Neutralization of errors and attacks in wireless ad hoc networks," in *International Conference on Dependable Systems and Networks*. IEEE, 2005, pp. 518–527.

[8] S. Yi and R. Kravets, "Key management for heterogeneous ad hoc wireless networks," in *IEEE International Conference on Network Protocols*. IEEE, 2002, pp. 202–203.

[9] B. Wu, J. Wu, E. B. Fernandez, M. Ilyas, and S. Magliveras, "Secure and efficient key management in mobile ad hoc networks," *Journal of Network and Computer Applications*, vol. 30, no. 3, pp. 937–954, 2007.

[10] J. Kong, H. Luo, K. Xu, D. L. Gu, M. Gerla, and S. Lu, "Adaptive security for multilevel ad hoc networks," *Wireless Communications and Mobile Computing*, vol. 2, no. 5, pp. 533–547, 2002.

[11] H. Luo, J. Kong, P. Zerfos, S. Lu, and L. Zhang, "Ursa: ubiquitous and robust access control for mobile ad hoc networks," *IEEE/ACM Transactions on Networking (ToN)*, vol. 12, no. 6, pp. 1049–1063, 2004.

[12] P.-A. Fouque, G. Poupard, and J. Stern, "Sharing decryption in the context of voting or lotteries," in *Financial Cryptography*. Springer, 2001, pp. 90–104.

[13] C. Cachin and A. Samar, "Secure distributed dns," in *International Conference on Dependable Systems and Networks*. IEEE, 2004, pp. 423–432.

[14] J. Bau and J. C. Mitchell, "A security evaluation of dnssec with nsec3," in *NDSS*. The Internet Society, 2010.

[15] C. Deccio, J. Sedayao, K. Kant, and P. Mohapatra, "Quantifying and improving dnssec availability," in *International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2011, pp. 1–7.

[16] A. Kasabov and Y. Schaeffer, "Resilient opendnssec," Universiteit van Amsterdam - NLnetLabs, The Netherlands, Tech. Rep., August 2012.

[17] D. B. Terry, M. Painter, D. W. Riggle, and S. Zhou, *The berkeley internet name domain server*. University of California, 1984.

[18] D. Kozic, B. Zwittnig, J. Sterle, and A. Kos, "Dnssec key management," *Elektrotehniski Vestnik/Electrotechnical Review*, vol. 79, 2012.

[19] H. Yang, E. Osterweil, D. Massey, S. Lu, and L. Zhang, "Deploying cryptography in internet-scale systems: A case study on dnssec," *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 5, pp. 656–669, 2011.

[20] J. Bustos-Jiménez, R. Alonso, C. Faúndez, and H. Méric, "Boxing experience: Measuring qos and qoe of multimedia streaming using ns3, lxc and vlc," in *Local Computer Networks Workshops (LCN Workshops), 2014 IEEE 39th Conference on*. IEEE, 2014, pp. 658–662.