Article can be accessed online at http: //www.publishingindia.com

A Zero Knowledge Proof Implementation Using NXP Security Engine

Alexandru Porosanu*, Nitesh Lal Narayan**

Abstract

Zero-knowledge proof is a protocol which does not reveal the any user information or secret during the authentication process. In this paper an efficient yet secure approach is proposed which could be used to for authenticating any user by using Digital Signature Algorithm and NXP security engine without exposing any of the user's secret information on the network.

Keyword : Zero Knolwedge Proof, Security Engine, Crypto Hardware Accelerator, Authentication

1. Introduction

Zero Knowledge, introduced in [1] formally is an interactive two party protocol, in which Prover is able to convince Verifier who follows the same protocol, by the overwhelming statistical evidence, it is a valid entity. In additions, during interactions, Prover does not reveal to Verifier any other information, except whether a given condition is true or not. Consequently, whatever Verifier can do after he gets convinced, he can do just believing that the condition is true.

Digital signature [2] can be verified by using the public key of the signing entity. The two most popular public-key algorithms which can provide digital signatures are RSA-type signature scheme [3], the security of which is based on factoring and the other is the ElGamal-type signature scheme [4], the security of which is based on the discrete logarithm problem over the finite field GF (p).

SEC is the chip's security engine, which serves as NXP's latest cryptographic acceleration and offloading hardware. It combines functions previously implemented in separate modules to create a modular and scalable acceleration and assurance engine. It also implements block encryption algorithms, stream cipher algorithms, hashing algorithms, public key algorithms, run-time integrity checking, and a hardware random number generator. SEC performs higher-level cryptographic operations than previous NXP cryptographic accelerators.

In this paper a solution is proposed a solution to establish a secure and efficient system for the verification of a client based on zero knowledge proof protocol using features provided by NXP Power PC platforms such as one-time programmable master key OTPMK [3] and hardware random number generator.

2. Zero Knowledge Proof

Zero-knowledge protocols allow identification, key exchange and other basic cryptographic operations to be implemented without leaking any secret information during the conversation and with smaller computational requirements than using comparable public key protocols. Thus Zero-knowledge protocols seem very attractive especially in smart card and embedded applications.

A. Entities in Zero Knowledge

A zero-knowledge proof is a two-entity protocol between a prover and a verifier such that it allows the prover to convince the verifier that he knows a secret value that satisfies a given relation, without the verifier being able

* NXP Semiconductors Romania Romania. Email: alexandru.porosanu@nxp.com
** Freescale Semiconductors India Pvt Ltd, India. Email: niteshnarayanlal@freescale.com,

to learn anything about the secret. We could explain this by taking a classic example [4] from Peggy (the prover of the statement) and Victor (the verifier of the statement).

In this story, Peggy has uncovered the secret word used to open a magic door in a cave. The cave is shaped like a ring, with the entrance on one side and the magic door blocking the opposite side. Victor wants to know whether Peggy knows the secret word; but Peggy, being a very private person, does not want to reveal the fact of her knowledge to the world in general.

They label the left and right paths from the entrance A and B. First, Victor waits outside the cave as Peggy goes in. Peggy takes either path A or B; Victor is not allowed to see which path she takes. Then, Victor enters the cave and shouts the name of the path he wants her to use to return, either A or B, chosen at random. Providing she really does know the magic word, this is easy: she opens the door, if necessary, and returns along the desired path.

However, suppose she did not know the word. Then, she would only be able to return by the named path if Victor were to give the name of the same path by which she had entered. Since Victor would choose A or B at random, she would have a 50% chance of guessing correctly. If they were to repeat this trick many times, say 20 times in a row, her chance of successfully anticipating all of Victor's requests would become vanishingly small (about one in 1.05 million).

Thus, if Peggy repeatedly appears at the exit Victor names, he can conclude that it is very probable - astronomically probable - that Peggy does in fact know the secret word.

One side note with respect to third party observers: Even if Victor is wearing a hidden camera that records the whole transaction, the only thing the camera will record is in one case Victor shouting "A!" and Peggy appearing at A or in the other case Victor shouting "B!" and Peggy appearing at B. A recording of this type would be trivial for any two people to fake (requiring only that Peggy and Victor agree beforehand on the sequence of A's and B's that Victor will shout). Such a recording will certainly never be convincing to anyone but the original participants. In fact, even a person who was present as an observer *at the original experiment* would be unconvinced, since Victor and Peggy might have orchestrated the whole "experiment" from start to finish.

Further notice that if Victor chooses his A's and B's by flipping a coin on-camera, this protocol loses its zero-knowledge property; the on-camera coin flip would probably be convincing to any person watching the recording later. However, digital cryptography generally "flips coins" by relying on a pseudo-random number generator, which is akin to a coin with a fixed pattern of heads and tails known only to the coin's owner. If Victor's coin behaved this way, then again it would be possible for Victor and Peggy to have faked the "experiment".

B. What Makes It 'Zero Knowledge'?

However how we could be so sure that no information is been leaked if this protocol is been followed. The first rule of modern cryptography is *never to trust people* who claim such things without proof. Goldwasser, Micali and Rackoff proposed three following properties that every zero-knowledge protocol must satisfy. Stated informally, they are:

1. *Completeness:* it states that if the statement is true, the honest verifier will be convinced of this fact by an honest prover.
2. *Soundness:* It states that if the statement is false, no cheating prover can convince the honest verifier that it is true, except with some small probability
3. *Zero-knowledgeness* : It states if the statement is true, no cheating verifier learns anything other than this fact.

3. Turning Zero Knowledge into Digital Signature

It is also possible to turn (generically) any zero-knowledge proof into a *digital signature* scheme. The idea is that in a ZK proof, the prover sends *commitments*, then the verifier sends a *challenge* to which the prover must respond; the involved mathematics ensure that a sham prover would not be able to respond correctly to all possible challenges confronted. What makes the proof "convincing" is that the verifier does not collude with the prover and really sends a completely random challenge. Thus, the ZK *interactive* proof is convincing only insofar as we trust the verifier for not colluding. The proof is turned non-interactive by computing the challenge with a hash function over the whole set of commitments from the prover (hash functions act "randomly" by construction). By including

the message to sign in the hash function input, you get a digital signature scheme.

4. Using NXP Platform Features

SEC provides platform assurance by working with SecMon, which is a companion logic block that tracks the security state of the chip. SEC is programmed by means of descriptors that indicate the operations to be performed and that point to the message and associated data. SEC incorporates a DMA engine to fetch the descriptors, read the message data, and write the results of the operations. The DMA engine provides a scatter/gather capability so that SEC can read and write data scattered in memory. SEC may be configured by means of software for dynamic changes in byte ordering. The default configuration for this version of SEC is big-endian mode.

4.1 SEC Random Number Generator

The RNG generates cryptographically-strong, random data. SEC's RNG utilizes a true random-number generator (TRNG) as well as a pseudo random-number generator (PRNG) to achieve both true randomness and cryptographic strength. The random numbers generated by the RNG are intended for direct use as secret keys, per-message secrets, random challenges, and other similar quantities used in cryptographic algorithms.

To enable support of SEC hardware random number generator, it must be enabled via kernel menuconfig option as shown in figure 1, on NXP platform (the example below is based on T1024 SoC). Once enabled, similar to the /dev/urandom special file, the special /dev/hwrng is created. Through this, an interface to the SEC's random number generator is provided to applications in user-space.

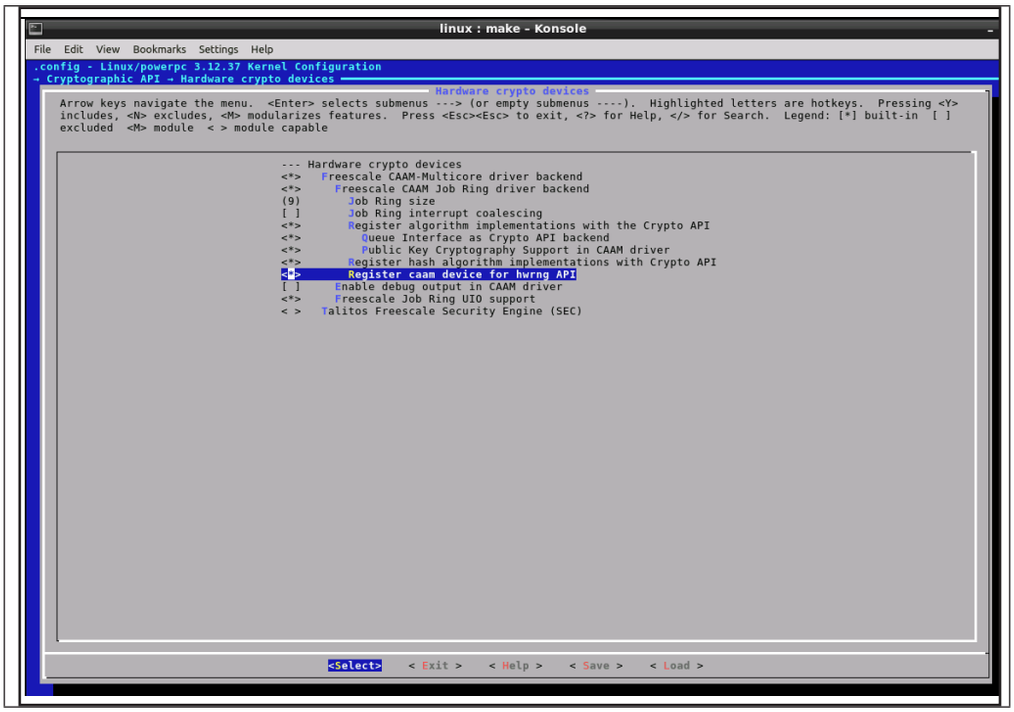

Fig. 1. Sample Menuconfig Screen Showing how to Enable SEC RNG

4.2 Using Pre-blown OTPMK (One Time Programmable Master Key)

The OTPMK registers are set by OEMs (Original equipment manufacturer) to configure a 256-bit secret value that becomes available to be used by the SEC module to derive the AES blob keys when the security monitor is in the trusted state or secure state. The 256-bit secret value is stored in a series of eight 32-bit registers OTPMKR0-OTPMKR7. The primary purpose of the

OTPMK is encryption and decryption of additional secret keys (also usable only by the SEC module) that can be used to protect arbitrary data. This level of indirection increases the difficulty of cryptanalysis of the OTPMK.

4.3 SEC Public Key HW module

The SEC hardware block offers the possibility of offloading the required public-key operations to be executed as part of this implementation, thus greatly speeding the verification process. In order to enable this offload, "Public Key Cryptography in CAAM" needs to be selected in the kernel configuration, as shown in the Figure 2.

Fig. 2. Sample menuconfig screen showing how to enable PKC in CAAM

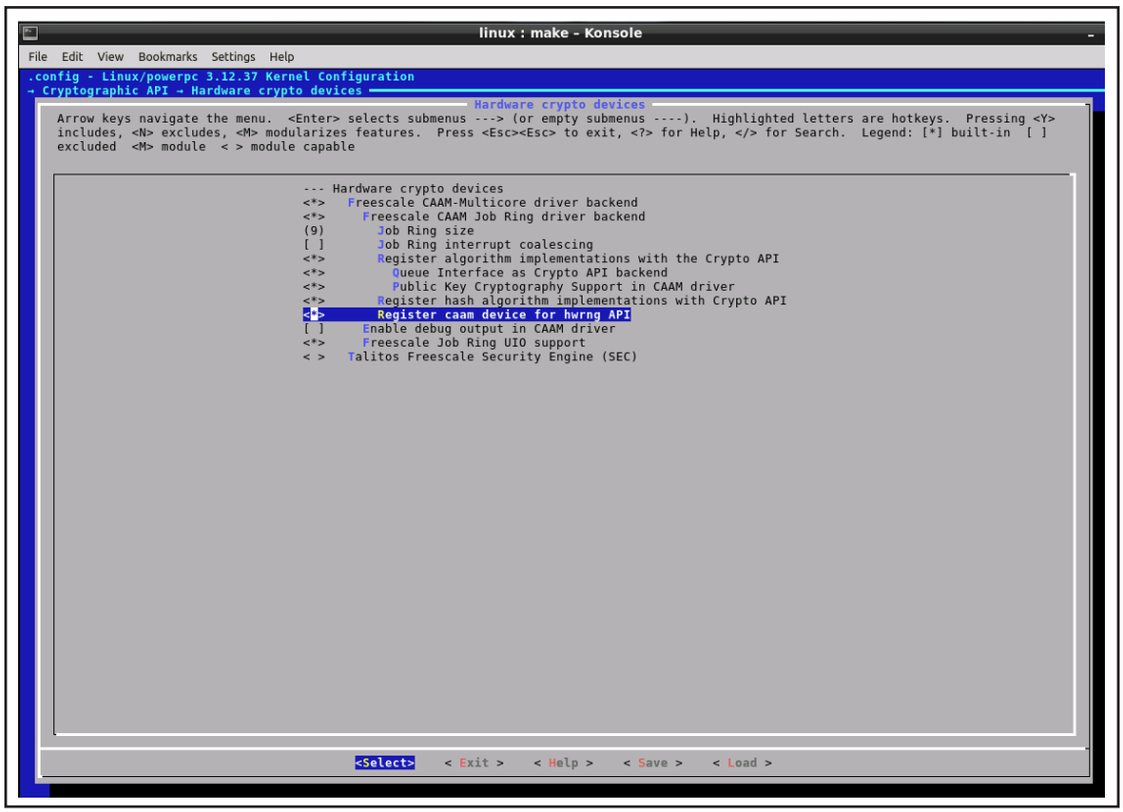

Fig. 1. Sample Menuconfig Screen Showing how to Enable PKC in CAAM

Furthermore, the public key module offers the possibility of performing compute intensive operations, for instance modular exponentiation, modular substraction/adition/ multiplication.

5. Proposed Scheme to Implement Zero Knowledge Proof using NXP Platform

In cryptography, zero-knowledge proof is primarily used as a means of entity authentication. That is, Peggy possesses some secret S that only she can know. She proves to Victor that she is indeed Peggy (and not an impostor) by proving that she possesses S. Of course, she wants to do so without revealing S to Victor (or any potential eavesdroppers).

The following implementation is based on the DSA signing algorithm. It's also assumed that following communication takes place after establishing a secure session with the server once it's verified that the server is authentic. The SEC module provides the necessary HW blocks to help with the necessary processing required for successfully implementing the ZPK:

- RNG block - for random number generation

- PKHA block - for DSA sign/verify functionality
- SecMon - for OTPMK-based key encryption/decryption without having the actual key disclosed to any other third party on either sides of the channel

The overall picture of the scheme proposed could be illustrated as per figure 3.

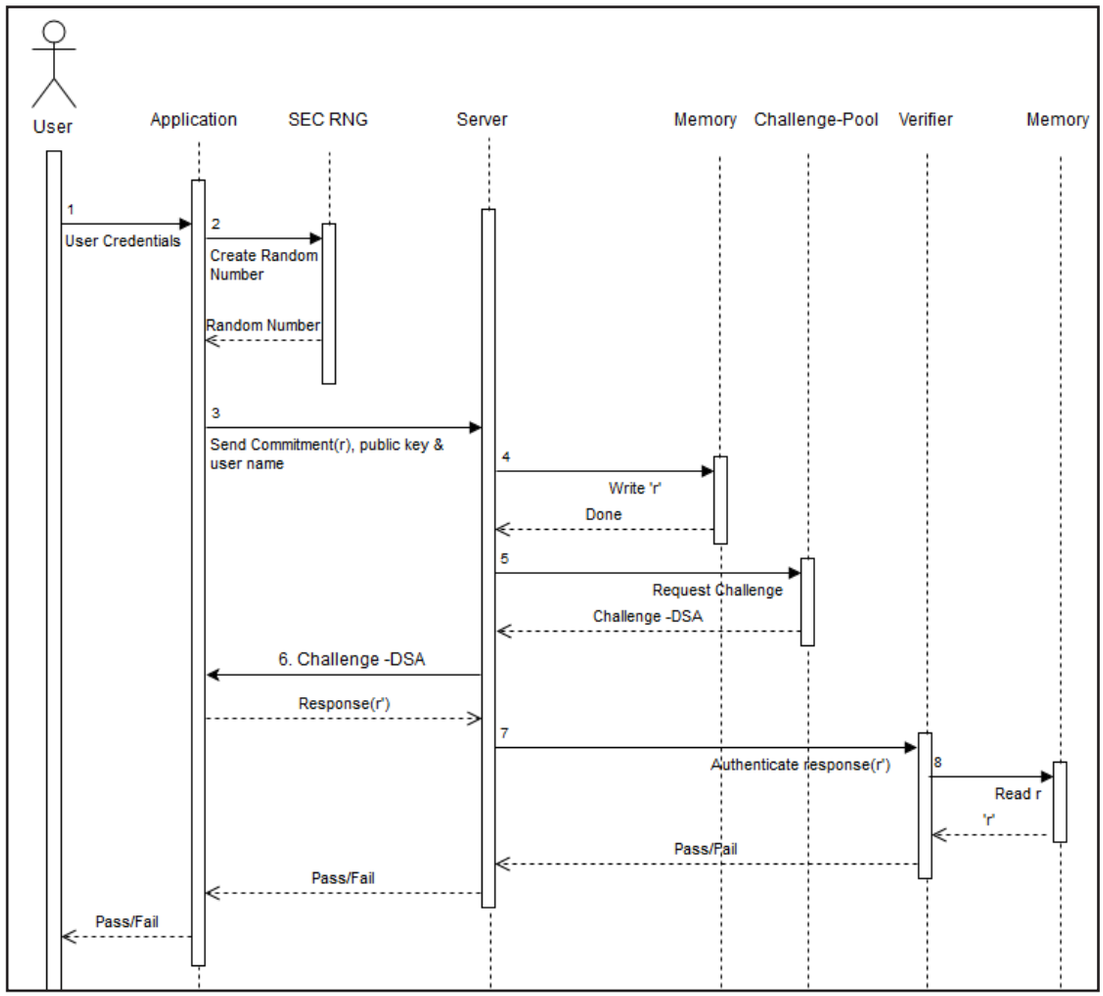

Fig. 3. Illustration of the Proposed Scheme

6. Implementation

Step 1 – DSA key pair generation

First a DSA key pair is generated. Firstly, the key length is selected, determined by L and N. According to FIPS 186-3 [1] pairs of values are (1024,160), (2048,224), (2048,256), and (3072,256).

Generating the DSA key pair can be done for instance using the OpenSSL application, which is a standard application that makes use of the SEC offered Public Key primitives.

The private part of the key, let it be called x is encrypted using a OTPMK derived key, when obtained from the SEC hardware block. Thus the user is not exposed at any time to the unencrypted private key.

The public key is composed of the quartet (p, q, g, y), where

- p is is a L-bit prime modulus such as p – 1 is a multiple of q
- q is a N-bit prime
- g is a number such as g^q = 1 (mod p)
- y = g^x mod p

The SEC hardware block is capable of generating in one step all the required values as well as the public & private keys.

Step 2 - Random Number Generation

Using /dev/hwrng special file (which makes use of the SEC hardware RNG block), generate a random number 'k', where 0 < k < q.

Step 3 - Signature Generation

This can be done by also using the OpenSSL application, which interfaces with the SEC Public Key hardware block. Let 'H' be the hashing function and 'm' the user name.

The message to be signed is H(m). Apart from this, the commitment value is calculated:

$$r = (g^k \bmod p) \bmod q$$

The values to be sent to the server are:

m - user name

(p, q, g, y) – public key

r – commitment value

Step 4 - Challenge Generation

After receiving the commitment value server will store it in r1 and will pose a challenge let us call it CHALLENGE-DSA. For now we are considering only one challenge to be imposed for simplicity. There could be a pool of challenges such as CHALLENGE-ECDSA and so on. This will be sent back to the client to after encrypting it with the public key (also shared by the client).

Step 5 - Response Calculation

As the client receives the CHALLENGE-X it will decrypt the request using the private key and then later calculate the response by using the private part i.e. response:

$$s = k^{-1}(H(m) + x * r) \bmod q.$$

This operation can be done only the original client because only that SEC block has the original OTPMK key which could decrypt the private key and use it.

This response will be sent back to the server.

Step 6 - Verification

After receiving the response, the server will attempt to verify the signature, by calculating the following values:

$$w = s^{-1} \bmod q$$

$$u1 = (H(m) * w) \bmod q$$

$$u2 = (r * w) \bmod q$$

$$r2 = ((g^{u1} * y^{u2}) \bmod p) \bmod q$$

If r1 = r2 then the client is verified. If the server is still not satisfied he could again send a challenge to the client.

These operations can be offloaded to the SEC Public Key block.

The overall diagram of the proposed scheme can be seen below:

7. Conclusion

An efficient scheme is presented in the paper. In our scheme, in order to authenticate a user to the server, it does not have to send its credentials but rather just need register its name and the public key part. The client may maintain public and private key pair depending on the NP-problem on which this protocol is based. Once that is done the server imposes certain challenges to the client,i.e., the client has to use the private key which is generated by using user's password. The proposed algorithm is not susceptible to all sort of man in the middle attack as no secret is been shared across the network.

References

1. Wang, H., & Sun, Z. (2013). Research on zero-knowledge proof protocol. *International Journal of Computer Science Issues*, January, 10(1), 194-200. Retrieved from http://ijcsi.org/papers/IJCSI-10-1-1-194-200.pdf
2. Dime W., & Hellman, M. (1976). *New directions in cryptography*. IEEE Transactions on Information Theory, 22(2), 644-654.
3. http://www.NXP.com/files/training_pdf/WBNR_FTF10_NET_F0695.pdf?lang_cd=en
4. Quisquater, J. J., Guillou, L. C., Berson, T. A. (1990). *How to explain zero-lnowledge protocols to your children.* Proceedings of Advances in Cryptology-CRYPTO '89, (pp. 628-631).
5. FIPS PUB 186-3: Digital Signature Standard (DSS). (2009). Retrieved from *csrc.nist.gov.*