HAL
open science

Privacy and Nomadic Computing: A Public-Key Cryptosystem Based on Passwords

Frederic Prost, Lydie Terras

To cite this version:

Frederic Prost, Lydie Terras. Privacy and Nomadic Computing: A Public-Key Cryptosystem Based on Passwords. 2013. hal-00933842

HAL Id: hal-00933842
https://hal.science/hal-00933842v1

Submitted on 21 Jan 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Les rapports de recherche du LIG

Privacy and Nomadic Computing
A Public-Key Cryptosystem Basedon Passwords

Frédéric PROST, Associate Professor, LIG, University of Grenoble (UJF), France
Lydie TERRAS, Master Student, LIG, University of Grenoble (UJF), France

RR-LIG-030
février 2013

http://rr.liglab.fr
ISSN 2105-0422

L I G

Privacy and Nomadic Computing
A Public-Key Cryptosystem Based on Passwords

Frédéric Prost[1], Lydie Terras[1]
[1] *Université de Grenoble – LIG, B. P. 53 - 38041 Grenoble Cedex 9 , France*
{*frederic.prost,lydie.terras*}*@imag.fr*

Keywords: Password, Public-key cryptography, Secure hashing functions, RSA, PGP

Abstract: The use of public-key cryptography is complicated in a nomadic computing era. Private keys are typically huge numbers that are impossible to memorize or even to write down and have to be stored electronically. Therefore a mobile user has somehow to keep its private key with him at all time (it is senseless to imagine that the private key is dowloaded through a public network). It is neither realistic nor safe. Indeed the mobile device must be protected, which is usually done through password mechanisms. At the end of the day all the cryptosystem relies on this password. In this paper we propose a generic way to produce keys from a password using secure hashing functions. We have done an implementation of a nomadic PGP as a proof of concept: the software is a java applet, thus platform independent, providing a complete solution for mail encryption based on public-key cryptography. In the end the user just has to remember its password, and no longer has to rely on specific software/operating system/hardware settings.

1 INTRODUCTION

Key management is a central concern in cryptosystems. Indeed, at the very core of any cryptosystem used for communication (when it is used to safely store information it is different) lies this paradox: the receiver should be able to decrypt the message, but can only do so with the appropriate key. Moreover this key has to remain secret and is linked to the key used to encrypt the message. Hence the question: if one is able to communicate safely encryption/decryption keys why does he need a cryptosystem in the first place? Instead of safely communicating the key it would be enough to safely transmit the message. Asymetric cryptography, like RSA (Rivest et al., 1978) or ElGamal (ElGamal, 1985), partially solves this paradox: the key used to encrypt data is public while the key used to decrypt remains private; moreover the relation between the keys is such that it is very hard (from a computational point of view) to compute the private key having only the knowledge of the public key (although it is easy if you have the suitable hints that remain private). The problem with such cryptosystems is that private keys are typically very big integers, typically fo a few thousand bits length, and are therefore impossible to remember or even (although it is not a good security policy) to write down in a scrapbook.

In our era of nomadic life and because of the multiplicity of means used to connect to the internet, the issue of how to carry your private key with you remains. Historically it was stored in one's personnal computer which was physically secured, under lock and key, at home or at the office. Nowadays people use more and more different ways and tools for internet and mail access: from smartphones to workstations through access with netbooks and random computers (think internet cafes, public computers in airport etc.).

In order to tackle with this problem we propose a system that produces keys from passwords. The raw idea is to have a deterministic, although non invertible, algorithm that relies on secure hashing functions. The result of the secure hashing function is used as a pseudo-random sieve to produce the keys. The system is implemented as a java applet and is platform independent: one just needs a browser, and also needs to remember its password, to generate and use a public key cryptosystem and no longer has to store its private keys.

We start in section 2 by a brief discussion of the security aspects related to this idea. In section 3 we present a method to generate keys for the RSA cryptosystem. We present a proof of concept in the form of a software that is a PGP-like software in which symetric and asymetric encryption are used together, called

Nomadic PGP in section 4. The basic blocks of Nomadic PGP are the hashing function Whrilpool, RSA for the asymetric encryption and twofish for the symetric encryption. Finally we conclude in section 5.

2 SECURITY DISCUSSION

As B. Schneier strengthens it again and again in (Schneier, 2003) all security problems/solutions are trade-offs. In essence we propose the use of a public key cryptosystem, in which the user may choose an arbitrary large security parameter, in a way that is completely determined by a password (and the length of the key to produce). Clearly since the aim of our system is to be able to keep the password in memory the key space is much smaller (if the password had the same length as the private key we would have gain nothing) than what it could theoretically be. By construction only a limited fraction of keys (regardless of the way keys are produced)can be explored by an implementation following our idea. The natural question is: how the restriction to passwords interfere with the strength of the original cryptosystem ? In other words what have we lost for the benefit of being able to remember a very large (the size of the generated key is unnbounded) private key ?

In order to fix ideas suppose that we are generating RSA keys of size 2048 bits (the size of the modulus) from passwords using a hashing function f. The raw idea is to produce enough blocks from the password in order to produce two primes of size 1024 by applying f to the password (with deterministic modifications of the password for each new block) and to multiply those two primes together (the detailed process is given in section 3).

The factorization of a 2048 bits integer is today out of reach: as of today the largest number factorised was RSA-768 of length 768 bits see (Kleinjung et al., 2010). So the easiest way to break the system is to recover the password using either a brute force attack, or some weakness of the hashing function f. If we rely on the assumption that f is a dependable secure hashing function (our system is completly parameterized by the hashing function used so it is a fair assumption) there only remains the question of the brute force attack. At this point we can note that the attack is more complicated than just computing a hash and comparing it to some stored value. Indeed the public value is computed from many hashes and is not instantaneous. We refer the reader to section 4 for practical results. Another important point to note is that numbers generated this way are pseudo-random, and uniformally distributed among numbers that are the product of two primes otherwise the security assumption on f would be wrong. In the last analysis the strengh of the system is the same as the strength of a secure symetric encryption algorithm using the password as an encryption key regardless of the size of the key generated.

From a more general point of view it seems that we loose nothing by limiting the strength of the cryptosystem to the strength of passwords. Indeed in every day life, even if the cryptosystem used is RSA with huge keys of length 4096 bits, then the easiest way to break the cryptosystem is not to try to factorize this number but to steal the private key that is stored on a computer or on some device. Even if the private key is encrypted it all boils down to find a password or two (one for login and the other one to decrypt the encrypted private key) to recover it in the vast majority of the cases.

3 RSA KEY GENERATIONS USING SECURE HASH FUNCTIONS

In our proof of concept we have used the Whirlpool hashing function, see (Barreto and Rijmen, 2003) for its formal definition, in order to generate a RSA key pair (public and private). In a nutshell (we refer to (Rivest et al., 1978) for the complete definition of the RSA cryptosystem) we need to produce to prime numbers p,q. Then let $n = pq$ and $\phi(n) = (p-1)(q-1)$. We need to choose e such that $\phi(n)$ and e are coprime and we have to compute d the multiplicative inverse of e modulo n. The public key is the pair e,n and the secret key is the pair d,n. In practice e is first chosen as the 5th Fermat number (if it doesn't work we deterministically chose the closest number coprime with n starting from it).

Whirlpool is a secure hashing function of the Miyaguchi-Preneel family and is based on AES (Daemen and Rijmen, 1999). Whirlpool produces a digest of size 512 bits for any message of lenght less than 2^{256} bits. The idea is to use the password digest as a sieve to look for a prime number: if the digest is a prime number it is ok otherwise we add 2 to the digest (if it is an odd digest of course) until we find the next prime number.

The first limitation that we have to overcome is the fixed length of the digest which is 512 bits. In order to look for bigger prime numbers in a deterministic way we are going to produce as many 512 bits blocks as necessary by applying repeatedly Whirlpool to the password. Since we do not want to repeat the same

block again and again we add the last bit of the previous to the password to compute the next block. We call this alogrithm WhirlpoolNBits. It is schematically presented on figure 1.

Since the RSA keys needs two prime numbers the process has to be repeated in order to create a second prime number. One choice is to concatenate the password with itself as a 'second password' sent to WhirlpoolNbits. This method can be used as often as necessary if more pseudo-random numbers are needed regarding the cryptosystem considered.

4 NOMADIC PGP

An applet based on this approach has been designed and implemented as a proof of concept. It is a nomadic version of PGP (Zimmermann, 1995) and can be used for mail encryption, decryption and authentication. It relies on RSA for the asymetric encryption and twofish, see (Schneier et al., 1999), for the symetric encryption. The idea of P. Zimermmann is to combine the best of both worlds: the asymetric encryption is used to encrypt a session key and the symetric encryption (much faster) use this session key to encrypt the message. Moreover, as in PGP, there is a signature mechanism for Nomadic PGP.

4.1 The nomadic PGP applet

The nomadic PGP applet provides three services:

- Key generation: a public key is generated from a password and a desired key lenght.
- Encryption of messages: with a message and a public key as input the applet produces an encrypted message that can be copy/paste as the mail to be sent. If the sender desires to sign its message he also has to give its password as input.
- Decryption of messages: The message to be decrypted and the password are given as input. If an authenticity check is need one also has to input the public key of the sender. The decrypted message together with the result of the authenticity check are displayed.

The Nomadic PGP can be downloaded from `http://membres-lig.imag.fr/prost/NPGP/`.

A typicaly use/case scenario is the following. Suppose that Alice wants to send an e-mail to Bob. Bob enters his password to generate a public key that he publishes on his website. Alice copies this public key and enters it with her password and the message she wants to encrypt in the application. It returns her the encrypted message. Alice copies and pastes the result on a e-mail client and sends it to Bob. Bob receives the e-mail, he copies and pastes the encrypted message in the application and enters also his password then it returns him the decrypted message (he can also gives the public key of Alice to check a signed message).

We briefly explain how each service of the applet is implemented in the following sections.

4.2 Encryption

The input for the encryption process consists in a public key, a message and a password. The first step is to generate the secret key as discussed in section 3. Then, a session key is generated as follows: the concatenation of the password, the message and a timestamp are hashed into a 256 bit digest using Whirlpool. This digest is the session key that is going to be used by twofish to encrypt the message. The output consists of three parts. The first part is the encryption by RSA of the session key with the public key that was given as input. The second part is the encryption by RSA of the pad size: since twofish is 128 bits block cypher it can be necessary to pad the original message with useless 0's, the pad size will allow the removal of this extra 0's after decryption. And finally the third part is the message encrypted with twofish and the session key. Note that because of the timestamp the session key is always different even for two mails with the same content.

The encryption process is schematically given in figure 2.

4.3 Decryption

The input for the decryption process consists in the password with a Nomadic PGP encrypted message. The firt step is to generate the private key associated with the password. This private key is used to decrpyt the first and second parts of the message to recover the session key and the pad size. The session key is given and the third part of the message are given as input to twofish, and the result of twofish is stripped off the unnecessary 0's thanks to the pad size.

The decryption process is given in figure 3.

4.4 Digital signature with nomadic PGP

Nomadic PGP includes a simple digital signature mechanism. During the encryption the user has to provide its password in order to generate its private key. Then a digest of the message is computed using Whirlpool and this digest is encrypted with RSA and the private key of the sender. The encrypted digest is

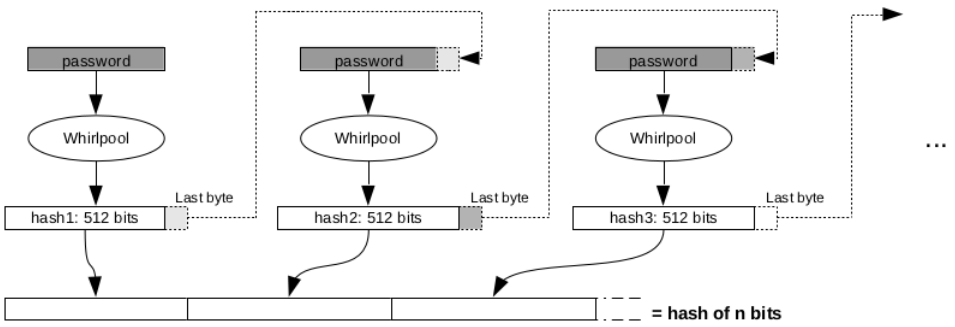

Figure 1: WhirlpoolNBits algorithm

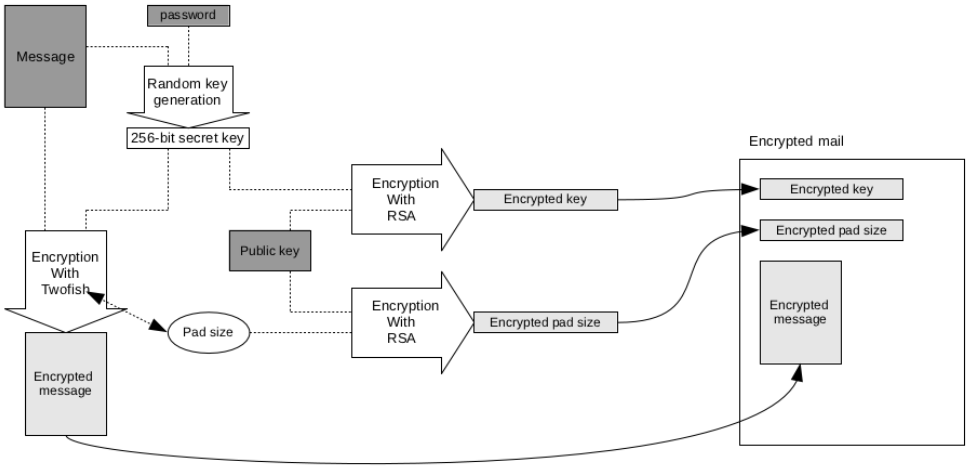

Figure 2: Encrytion process

appended to the message to be sent. On the receiving end, the message, the password of the recipient and the public key of the sender are needed. The decryption is performed as described in section 4.3. The public key of the sender is used to decrypt the last part of the message which is compare to the digest of the decrypted message. If it matches then the signature is valid.

The signature scenario is schematically depicted in figure 4.

5 CONCLUSION

We have proposed and implemented a nomadic PGP including a basic signature scheme. Instead of having to deal with a private key management (where to store the private key, how to protect it etc.), the user has only to remember a password in order to generate its private key and to cypher/decypher messages. Moreover, the key generation as well as the cryptographic system are implemented as a java applet. Thus Nomadic PGP is platform independant and can be used anywhere without requiring specific software environment. This provides a fully nomadic cryptographic solution.

From a security point of view it appears that the reduction of the strength of cryptographic systems to the strength of a, humanly memorizable, password is realistic. Indeed most platforms, especially mail servers, cloud services and so on are protected by password mechanisms and if a more complicated key is need it is most probably going to be stored in a place protected by a password. We think that the development of frameworks allowing detemernistic and non reversible generation of complicated keys (especially in the case of public cryptography and its uses for communication and certification) is the path to fol-

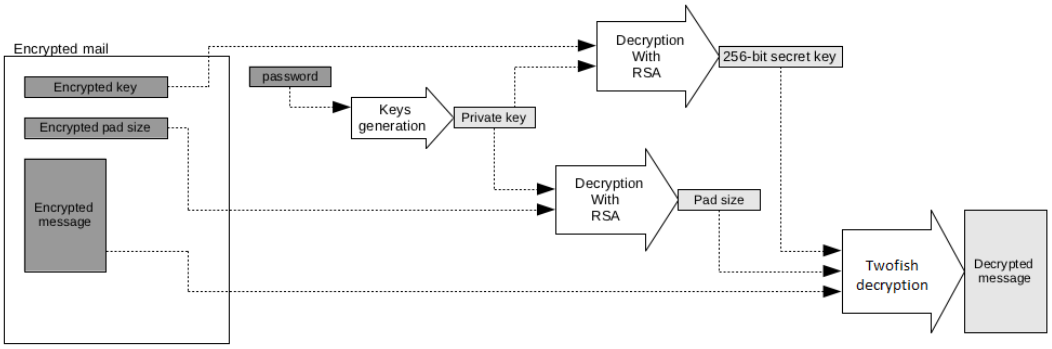

Figure 3: Decryption process

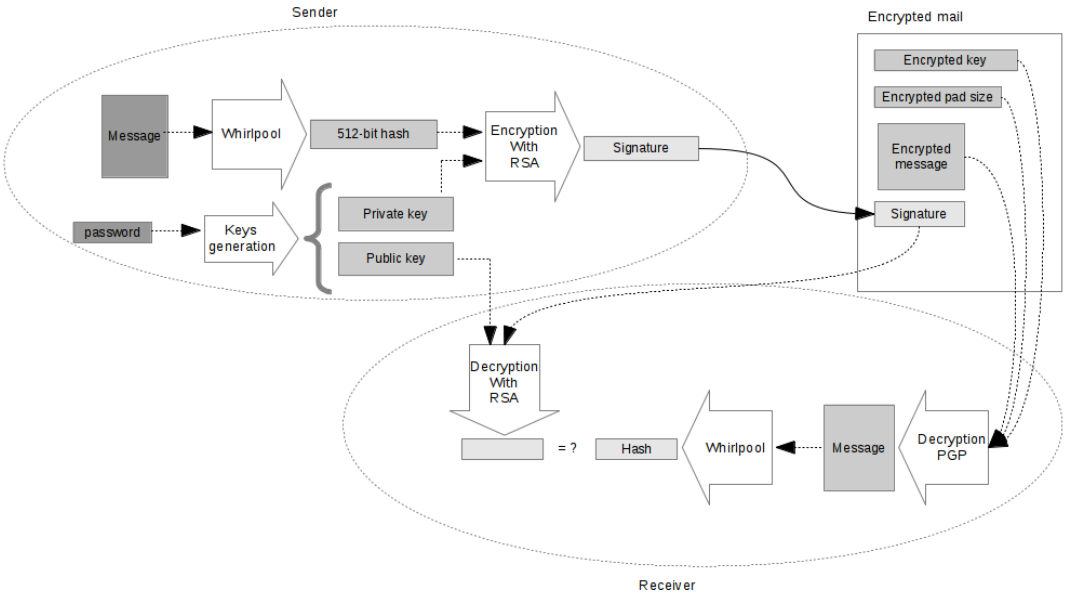

Figure 4: Signature process

low in a world where nomadic behavior takes more and more place in our every day life. An interesting field of application would be a Voice over IP solution based on those lines.

REFERENCES

Barreto, P. and Rijmen, V. (2003). The whirlpool hashing function. `http://www.larc.usp.br/~pbarreto/WhirlpoolPage.html`.

Daemen, J. and Rijmen, V. (1999). Aes proposal: Rijndael. AES Algorithm Submission. `http://www.cryptosoft.de/docs/Rijndael.pdf`.

ElGamal, T. (1985). A public-key cryptosystem and a signature scheme based on discrete logarithm. *IEEE Transactions on Information Theory*, 31(4).

Kleinjung, T., Aoki, K., Lenstra, A., Thomé, E., Bos, J., Gaudry, P., Kruppa, A., Montgomery, P., Osvik, D., Riele, H., Timofeev, A., and Zimmermann, P. (2010). Factorization of a 768-bit rsa modulus. Cryptology ePrint Archive, Report 2010/006. `http://eprint.iacr.org/`.

Rivest, R., Shamir, A., and Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126.

Schneier, B. (2003). *Beyond Fear. Thnking Sensibly about Security in an Uncertain World*. Springer Verlag.

Schneier, B., Kelsey, J., Whiting, D., Wagner, D., Hall, C., and Ferguson, N. (1999). *The Twofish Encryption Algorithm: A 128-Bit Block Cipher*. Wiley.

Zimmermann, P. (1995). *The Official PGP User's Guide*. MIT Press.

L I G