Trusted Cryptographic Tools Locking

V.N.Tsypyschev[1,2,3]

[1] Moscow Technological University, 78, Vernadsky Avenue, Moscow, 119454, Russian Federation
tsypyschev@yandex.ru
[2] Moscow Institute of Physics and Technologies (State University), 9, Institutsky pereulok, Dolgoprudny, Moscow Region, 141701, Russian Federation
[3] S-Terra CSP, 5, Georgievsky ave., Zelenograd, Moscow, Russian Federation

Abstract. Up to current moment it is actual to lock trustworthy cryptographic services of the complex product providing information security services. These cryptographic services, as usual, are included in as part of delivering complex product and must not be activated without proper additional license.
In this article, we provide a method of cryptographic services locking and a protocol of its activation including a legal customer authentication.
The main reason of this work is that the law of some countries prepends to export complex information security products with available cryptographic tools.

Keywords: Trustworthy Services Locking, Cryptographic Protocol, Software Customer Authentication

1 Introduction

The task concerning in this article is to provide a method of reliable, robust, secure, and solid locking of a part of software services, especially its cryptographic tools, until it'll be legally activated. A protocol of authentication of legal customer and services activation is a main part of this method.

Further we assume that

1. Cryptographic tools developer must to create and implement an algorithm of legal customer's license generation and validity checking under condition that software code may be publicly known and doesn't contain confidential data.
2. The algorithm of license generation and its validity checking must provide an unique license for given delivery and legal customer under condition that software code must be fixed and unaltered.
3. The algorithm of license validity checking must be executed successfully if and only if legal customer has typed his unique license data (LD) during installation process.
4. The algorithm of license validity checking must be implemented in an installer.

5. The algorithm of license validity checking must involve data inextricably linked with software code
6. Installer must activate cryptographic tools only after successful license data and legal customer data validity checking
7. It is technically impossible to activate cryptographic tools without installer.
8. It is technically impossible to simulate an installer to activate cryptographic tools
9. It is technically impossible to use an inactivated cryptographic tools code
10. For every legal customer the activation of cryptographic tools is executed by himself or by trusted employee.

So, cryptographic tools developer must be directed by paradigm "If not activated, then locked".

The difficulty of construction of such algorithm of generation license and its verification is consisted in the demand to type the license during the installation process. It means that the length of license couldn't be more than 25 bytes of printable characters.

Avoiding it we could go this evident way: to form the file from developer data, product custom code, legal customer data, license data and sign it by secret key fixed for whole custom. Then the verification algorithm should consist in signature verification using an open key inextricably linked to the cryptographic tools code.

Other way possible to use is to encipher those data by trusted two-key algorithm and further decipher it during installation process.

The inconvenience of those ways consists in that the signature should be at last 64 bytes of unprintable characters and after an ASCII85 application it should generate a license of at least 80 bytes. It is evident that so long license can't be typed during verification process.

Indeed, Bundesamt fuer Siecherheit in der Informationstechnik (BSI), Germany, has published [2, Table 3.2] this data:

Table 1. BSI estimation of two-key algorithms size and its complexity

$\lg_2 R$	ECDLP	Factorization / DLP in $\mathbb{F}_p^*$
60	120	700
70	140	1000
100	200	1900
128	256	3200
192	384	7900
256	512	15500

Here $ECDLP$ denotes the size of group of points of elliptic curve with complexity of finding discrete logarithm by Pollard's ρ-method is equal to 2^R group operations, $Factorization$ denotes the RSA-modulus size with complexity of factorization by number field sieve method equal to 2^R operations of ring $\mathbb{Z}/2^{Factorization}$, $DLP\ in\ \mathbb{F}_p^*$ denotes that the complexity of discrete logarithm problem in $\mathbb{F}_p^*$ is equal to 2^R operations of this field in terms of number field sieve method.

In this connection it is assumed that the parameters of elliptic curves are chosen such that the complexity of Menezes–Okamoto–Vanstone method is not less then the Pollard's ρ-method.

Besides that, the data of [2, Table 3.2] are coordinated to Post Quantum Cryptography problem.

According to [2, Table 3.2] neither one of well-known and trusted algorithms of [3, 4, 7, 8, 6], nor RSA, ECDSS, GOST R 34.0—2012 can be used to settle our demands.

So, the way could be used is to generate license as a result of pseudo–random function based on "good" cryptographic hash function as in [5] applied to that file . This way allows to obtain a license of acceptable length, but now to verify license it is needed to have a key of pseudo–random function what tends us to the conception of Trusted License Server which would provide a verification algorithm by needed secret information.

But now difficulties of communication with Trusted License Server arise.

Investigated all the reasons above we provide the method of licensing presented below.

2 Notation

1. E_K, D_K denote enciphering/deciphering in ECB mode with key K via block cipher (AES or Kuznechik (GOST R 34.12—2015))
2. $E_{K,IV}, D_{K,IV}$ denote enciphering/deciphering in CBC mode with key K and initial vector IV via block cipher (AES or Kuznechik (GOST R 34.12—2015))
3. $Imito_{K,IV}$ denotes calculation of CBC-MAC with key K and initial vector IV via block cipher (AES or Kuznechik (GOST R 34.12—2015))
4. $HASH$ denotes cryptographic hash-function (SHA or Streebog (GOST R 34.11—2012))
5. $HMAC(Key, Material)$ denotes pseudo-random function [5] with key KEY and publicly known $Material$ based on $HASH$-function
6. $Sign_K(M), VerSign_K(M)$ denote generation/verification of digital signature (ECDSS or GOST R 34.10—2012) with key K. If key is omitted then it's fixed and not altered
7. prime p is a modulus of elliptic curve
8. $E_{a;b}(\mathbb{F}_p)$ is an elliptic curve with its invariant $j(\mathcal{E}_{a;b})$ and coefficients $a; b \in \mathbb{F}_p$ such that
$$4a^3 + 27b^2 \not\equiv 0 \pmod{p},$$

9. integer m is an order of group of points of ellipic curve $E_{a;b}(\mathbb{F}_p)$;
10. prime q is an order of cyclic subgroup of points group of $\mathcal{E}$ such that

$$\begin{cases} m = nq; n \in \mathbb{Z}; n \geq 1; \\ 2^{254} < q < 2^{256} \end{cases}$$

11. $P \neq \mathcal{O}$ is a point of elliptic curve $\mathcal{E}$, $P = (x_p; y_p)$, such that $[q]P = \mathcal{O}$;

We suppose that the either American suite, either Russian suite of cryptographic primitives should be applied because of the compatibility reasons.

3 Method of cryptographic tool locking and license generation

Now let's assume that cryptographic tools code is locked via enciphering in CBC mode according to trusted block cipher (AES or Kuznechik) of some text file which contains some specific data. This data is an input of some specific calculation. The output this calculation determines work capability of the cryptographic tools.

Enciphering must be executed in CBC mode with initial vector IV and with key $K = x_C$, where x_C is a x-coordinate of point $C = [k]P$ of elliptic curve $\mathcal{E}_{a,b}$, $\operatorname{ord} P = q$.

As an example of the data we provide sample $Y, K, Sign(Y, K)$.

As an example of specific calculation we provide:

1. verification $VerSign(Y, K)$ with an inseparably tied with cryptotool code signature verification key, fixed for all version of cryptotool
2. calculation of
$$X = D_K(Y),$$
3. calculation of
$$H = HASH(X),$$

and after that H is essentially linked in cryptographic calculation process.

Further,

1. License structure has the form:

License :
$License := \{\ CodeOfProduct;$
$NumberOfLicense;$
$CodeOfCustomer;$
$LicenseCode;\}$

Here $CodeOfProduct$ is a vendor's code of cryptotool software, $NumberOfLicense$ is a number of issued license. Herewith vendor takes into account all issued licenses. $CodeOfCustomer$ is a setting data of the legitimate purchaser.

$LicenseCode$ is the result of application injective publicly-known and invertible code:

$$LicenseCode = EncodingAlgorithm(InitialLicenseCode).$$

As examples of such codes Base64 and ASCII85 serves.
$InitialLicenseCode$ is an output of pseudo-random generator of the length $N_1 = 8n$ bit.
After encoding length of field $LicenseCode$ is a α times greater:

$$N_2 = |LicenseCode| = \alpha \cdot |InitialLicenseCode| = 8\alpha n.$$

2. Further we shall represent fields (bit sequences) LicenseCode, NumberOfLicense and others as an integers where the most left bit of the field matches to most significant bit of the integer
3. Trusted License Server stores all issued licenses identifying it by CodeOfProduct and NumberOfLicense
4. Trusted License Server is able to generate digital signature. Verification key of this signature is common for te cryptotool type, non-alter allthe time of existence of cryptotool type, and inseparably linked with cryptotool code
5. The cryptotool activation performs by built-in installer. Installer is able to perform functions of digital signature verification, cryptographic hashing, and enciphering/deciphering. All thi functions are from the same suite of cryptographic primitives.
6. Cryptographic properties of installer can't be modified by user
7. Cryptographic properties of installer are developed to be executed by user without essential support of developer
8. Technical documentation (technical description of cryptographic suite, intermediary protocols, interface description, etc.) is publicly accessible
9. Parameter $CodeOfProduct$ is inseparably linked in code of installer

4 License validity verification protocol

Just now verification of license validity and activation of cryptotool code perform according to this

Protocol

1. Installer gets from keyboard license number N, purchaser's code C, and license code L.
   Meanwhile installer gets timer's indications in moments of keyboard clicking and stores it as sequence $(T_1, \ldots, T_M)$.
2. Installer forms pseudo-random values:

$$Nonce = HASH(T_1 \oplus T_3 | \ldots | T_{M-2} \oplus T_M);$$

$$Nonce_1 = HMAC(Nonce, N|L);$$

$$Nonce_2 = HMAC(Nonce, Nonce_1|N|L);$$

3. Installer forms and sends to Trusted License Server request:

$$(CodeOfProduct, N, [Nonce_2 \pmod q]P).$$

4. Trusted License Server forms and sends to installer reply

$$((CodeOfProduct, N, [Y]P), Sign(CodeOfProduct, N, [Y]P)),$$

herewith Y is a output of pseudo-random generator formed as an integer modulus q

5. Installer verifies

$$Sign(CodeOfProduct, N, [Y]P),$$

and forms seance key

$$SeanceKey = HASH([Nonce_2 \pmod q][Y]P).$$

If $Sign(CodeOfProduct, N, [Y]P)$ is not correct, then installer informs Trusted License Server, sends it pair $Nonce, Nonce_1$ and stops activation.

6. If Trusted License Server is informed by installer about stop of activation and get $Nonce, Nonce_1$, it verifies that

$$[Nonce_2 \pmod q]P = [HMAC(Nonce, Nonce_1|N|L) \pmod q]P.$$

If it is, it stops activation too and increase counter of activation tries. In other case it proceeds.

7. Trusted License Server forms seance key:

$$SeanceKey = HASH([Y][Nonce_2 \pmod q]P).$$

8. Trusted License Server gets an output of pseudo-random generator and forms $InitialVector$ for CBC mode and $ImitoKey$ to form CBC-MAC
After that it sends to installer

$$(NewInitialVector, NewImitoKey) = E_{SeanceKey}(InitialVector, ImitoKey),$$

herewith enciphering is performed in ECB mode.

9. Installer sends to Trusted License Server values

$$(NewL, ImitoL) = \\ (E_{SeanceKey,InitialVector}(L), Imito_{ImitoKey,InitialVector}(L)),$$

herewith enciphering is performed in CBC mode with $InitialVector$ as an initial vector, and

$$ImitoL = Imito_{ImitoKey,InitialVector}(L)$$

is a CBC-MAc with key $ImitoKey$ and initial vector $IiniialVector$ for $Nonce_1$

10. Trusted License Server calculate

$$L = D_{SeanceKey,InitialVector}(NewL),$$

and calculate its CBC-MAC.
In the case if calculated CBC-MAC of L differs from received CBC-MAC of L then Trusted License Server stop activation process, sends to installer notification signed with its secret key.
Trusted License Server increase the counter of activation tries.
In other case Trusted License Server verifies that

$$L = LicenseCode$$

stored at Trusted License Server Data Base. If it is, Trusted License Server proceed.
Else Trusted License Server stop activation process, sends to installer notification signed with its secret key.
Trusted License Server increase the counter of activation tries.
11. Installer sends to Trusted License Server values

$$(NewNonce_1, ImitoNonce_1) = \\ (E_{SeanceKey,InitialVector}(Nonce_1), Imito_{ImitoKey,InitialVector}(Nonce_1)),$$

herewith enciphering is performed in CBC mode with $InitialVector$ as an initial vector, and

$$ImitoNonce_1 = Imito_{ImitoKey,InitialVector}(Nonce_1)$$

is a CBC-MAc with key $ImitoKey$ and initial vector $IiniialVector$ for $Nonce_1$
12. Trusted License Server calculate

$$Nonce_1 = D_{SeanceKey,InitialVector}(NewNonce_1),$$

and calculate its CBC-MAC.
In the case if calculated CBC-MAC of $Nonce_1$ differs from received CBC-MAC of $Nonce_1$ then Trusted License Server stop activation process, sends to installer notification signed with its secret key.
Trusted License Server increase the counter of activation tries.
In other case Trusted License Server calculates

$$X_1 = ((Nonce_1 \oplus (LicenseCode|10\ldots0)) \pmod q)^{-1} \pmod q,$$

computes

$$Key = [X_1][k]P,$$

and sends to installer

$$(EncryptedKey, EncryptedIV, Imito) = \\ (E_{SeanceKey}(Key, IV), Imito_{ImitoKey,InitialVector}(Key, IV)).$$

13. Installer receives $(EncryptedKey, EncryptedIV)$, computes in ECB mode

$$Key = D_{SeanceKey}(EncryptedKey),$$

$$IV = D_{SeanceKey}(EncryptedIV),$$

and check CBC-MAC.
In the case if received CBC-MAC differs from calculated CBC-MAC installer informs Trusted License Server, protecting the notification by CBC-MAC, computed with $SeanceKey, InitialVector$. After that installer stops activation.
Trusted License Server increase counter of activation tries.
In other case installer represents Key as a point Q of elliptic curve $\mathcal{E}_{a,b}$ and computes

$$C = [(Nonce_1 \oplus (L||10\dots0)) \pmod q]Q$$

After that installer deciphers with key $K = x_C$ and initial vector IV enciphered in CBC mode text file and after that start installation.
If installation succeed then Installer informs Trusted License Server and protects notification by CBC-MAC calculated with $SeanceKey$, $InitialVector$.
14. Trusted License Server set $InstallationFlag$ to 1 for pair N, L.

End of Protocol

Note 1. Trustworthy of this Protocol may be increased if installer provided by VPN-client, pre-determinately configured to connect Trusted License Server only

5 Security considerations

In [1] were formulated goals to be achieved in cryptographic protocol construction process whether if those applicable to considered protocol. Below we list goals achieved in considering Protocol.

1. G1 (peer entity authentication)
2. G2 (message authentication)
3. G3 (replay protection in the sense that all messages were generated in current seance)
4. G7 (key authentication)
5. G8 (key confirmation, key proof of possession)
6. G9 (perfect forward secrecy)
7. G10 (fresh key derivation)
8. G12 (confidentiality, secrecy)
9. G16 (sender invariance)

6 Conclusions

So we have proposed the method of trusted cryptographic tools locking, its licensing, and protocol for license verification and cryptographic tools activation

Also we have provided some consideration about trustworthy of Protocol proposed

References

1. Automated Validation of Internet Security Protocols and Applications (AVISPA). IST-2001-39252. Deliverable D6.1 List of Selected Problems. 2005. http://www.avispa-project.org/publications.html
2. Cryptographic Mechanisms: Recommendations and Key Lengths // Bundesamt fuer Siecherheit in der Informationstechnik, BSI TR-02102-1, February 22, 2017
3. Buchmann J., Dahmen E., Huelsing A. (2011) XMSS - A Practical Forward Secure Signature Scheme Based on Minimal Security Assumptions. In: Yang BY. (eds) Post-Quantum Cryptography. PQCrypto 2011. Lecture Notes in Computer Science, vol 7071. Springer, Berlin, Heidelberg
4. ElGamal T. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms// Advances in Cryptology - Crypto 84. Springer. 1984. pp. 10-18.
5. D. Harkins, D. Carrel The Internet Key Exchange (IKE) // Request for Comments: 2409, https://tools.ietf.org/html/rfc2409
6. KCDSA Task Force Team The Korean Certificate-based Digital Signature Algorithm // http://grouper.ieee.org/groups/1363/P1363a/contributions/kcdsa1363.pdf see also ISO/IEC 15946-2
7. Kaisa Nyberg, Rainer Rueppel A New Signature Scheme Based on the DSA Giving Message Recovery // 1st ACM Conference on Computer and Communication Security. ACM Press. 1993. pp. 58 61. 372
8. C.P. Schnorr (1990), "Efficient identification and signatures for smart cards", in G. Brassard, ed. Advances in CryptologyCrypto '89, 239-252, Springer-Verlag. Lecture Notes in Computer Science, nr 435