

# Chapter 14: YYPKS: A Simple Public Key Stegosystem\*

Adam L. Young and Moti M. Yung

## Abstract

We present YYPKS, an experimental public key stegosystem (PKS) that we designed and implemented. YYPKS satisfies the basic properties of a public key cryptosystem. However, the ciphertexts that are output have the property of being uniformly pseudorandom bytes. We do not provide a formal proof of security for YYPKS, although we apply sound paradigms such as the *probabilistic bias removal method* (PBRM) to achieve pseudorandomness. Our goal was to develop a simple and practical PKS with security that can be argued on heuristic grounds. The experimental implementation `yypks.exe` is a command-line program that uses standard X.509 digital certificates and PKCS #12 files for RSA encryption and decryption, respectively. We hope that this fundamental research in cryptology will encourage further research and experimentation in the area of public key steganography.

**Key words:** Steganology, stegano, stegotext, cryptology, ciphertext, plaintext, cleartext, probabilistic bias removal method, RSA, OpenSSL.

## 1 Introduction

A public key stegosystem enables Alice to send an asymmetric ciphertext to Bob using a traditional public key infrastructure. It has the added property that ciphertexts appear to all efficient algorithms like a random stream of bytes.<sup>1</sup> A PKS may serve as an intermediate step in an overall communication protocol. For example, a plaintext may be encrypted using a PKS and

---

\*If this file was obtained from a publicly accessible website other than the website [www.cryptovirology.com](http://www.cryptovirology.com) then (1) the entity or entities that made it available are in violation of our copyright and (2) the contents of this file should therefore not be trusted. Please obtain the latest version directly from the official Cryptovirology Labs website at: <http://www.cryptovirology.com>.

<sup>1</sup>When the private decryption key is private, of course.

the resulting bit stream may then be steganographically encoded into some form of media (e.g., sound/video).

In this chapter we present a hybrid PKS that employs the RSA algorithm [10] and the IDEA block cipher [3, 4] in cipher block chaining mode (CBC). We designed YYPKS based on the probabilistic bias removal method [14] that we originally devised to mount a kleptographic attack on RSA key generation.

The PKS is by no means a typical implementation of a hybrid cryptosystem. Encryption is probabilistic and has a non-zero yet negligible failure probability. PKS encryption exhibits a feature of Las Vegas algorithms. In theory it is possible that a particular invocation of PKS encryption will never terminate. However, this should absolutely never happen in practice. In short, coin flipping is used to ensure that the asymmetric ciphertexts appear as uniformly random bit strings as opposed to always being members of  $\mathbb{Z}_n^*$ .

Our goal was to achieve an effective PKS that adheres to industry standards (PKCS #1, PKCS # 5, PKCS #12, X.509) and that is heuristically sound. We emphasize that no formal proof of security is provided. For example, we make the implicit assumption that a CBC ciphertext computed using IDEA “looks random”. To achieve this pseudorandomness, YYPKS encryption generates an IDEA key and initialization vector uniformly at random and asymmetrically encrypts both.

In support of our approach we provide a concise description of the YYPKS encryption and decryption algorithms in Section 3. As a result researchers need not “reverse-engineer” the algorithms from the ANSI C source code in the appendix. We encourage researchers and students to analyze YYPKS, break it, fix it, and experiment with it. Feedback to Cryptovirology Labs is welcome.

## 2 Background

The specific problem of implementing steganographic protocols in a public key setting was introduced in [13]. In particular Ahn and Hopper presented an RSA-based public key stegosystem (the paper employs the PBRM). Related work includes [2]. An elliptic curve public key stegosystem that uses twisted elliptic curves was presented by Möller [5]. Recent work that accounts for active attacks includes [1]. We recommend these papers as a starting point for an in-depth study of public key steganography.

**String Operations:** Let  $A$  and  $B$  be bit strings.  $|A|$  denotes the length in bits of  $A$ . Provided  $|A| = |B|$ ,  $A \oplus B$  denotes the bitwise logical exclusive-or of  $A$  with  $B$ .  $A || B$  denotes the concatenation of  $A$  with  $B$ . The function  $\text{SHA512}(A)$  returns the SHA-512 [6] hash of  $A$ . The function  $\text{SHA1}(A)$  returns the SHA1 hash of  $A$ . Define  $A \gg b$  to be the string that results from right shifting  $A$  by  $b$  bits. For example,  $1011 \gg 2 = 10$ .

**Other Notation:**  $r \in_R S$  denotes choosing element  $r$  uniformly at random from set  $S$ .  $\mathbf{Z}_n$  denotes the set of integers between 0 and  $n - 1$ , inclusive. Let  $\mathbf{Z}_n^*$  denote the set of integers in  $\mathbf{Z}_n$  that are relatively prime to  $n$ .

In this chapter we are careful to treat binary strings as objects that are separate and distinct from integers. Crypto papers often pass integers as input to hash functions, which is hazardous. The issue has to do with the possible existence of leading zeros. For example, when  $113 = 0x71$  is hashed, is it correct to write “SHA1(113)”? We argue that this is confusing since it could mean “SHA1(1110001)” if it is passed as a bit stream or it could mean “SHA1(01110001)” when it is passed as a single byte.

Also, we use logical indentation to show the body of loops, if statements, and so on. We chose to do this rather than enclosing statements in, say, begin and end.

### 3 YYPKS

The asymmetric encryption algorithm outputs a sequence of pseudorandom ciphertext bytes. In our design of YYPKS encryption we make the simplifying requirement that  $n$  and the plaintext are both a multiple of 8 bits in length. The predicate `IsAcceptableMod` is used to verify that  $n$  is an acceptable modulus for YYPKS encryption.

`IsAcceptableMod( $n$ ):`

Input: RSA modulus  $n$

Output: 1 if  $n$  is acceptable, 0 otherwise

1. let  $n_s$  be  $n$  represented as a binary string
2. if  $|n_s| < 1024$  output 0 and halt
3. if  $|n_s| \bmod 8$  is not 0 then output 0 and halt
4. output 1

`YYPKSEncr` is an asymmetric encryption algorithm. It encrypts the 352-bit message  $m_s$  using the recipient’s RSA public key  $(n, e)$ . The SHA1 hash

algorithm is used to implement a cryptographic integrity check on  $m_s$ . This provides heuristic *message-awareness* and *non-malleability*.

YYPKSEncr( $m_s, n, e$ ):

Input: plaintext bit string  $m_s \in \{0, 1\}^{352}$ ,

RSA modulus  $n$  and corresponding RSA public exponent  $e$

Output: result code  $\tau \in \{0, 1\}$  and asymmetric stegotext  $c_s$

1. let  $n_s$  be  $n$  represented as a binary string and set  $c_s = 0^{|n_s|+512}$
2. if (IsAcceptableMod( $n$ ) = 0) then
3.     print “bad modulus n”, set  $\tau = 0$ , and goto step 19
4. set  $max = 2^{|n_s|}$  and set  $L = max - n$
5. choose  $z \in_R \mathbb{Z}_n$  and choose  $b \in_R \{0, 1\}$
6. if  $z \in \mathbb{Z}_n^*$  then
7.     choose  $r \in_R \mathbb{Z}_n^*$  and compute  $s = r^e \bmod n$
8.     let  $r_s$  be  $r$  represented as a binary string with leading zeros added as needed to make  $|r_s| = |n_s|$  and set  $pad = \text{SHA512}(r_s)$
9. else
10.    set  $s = z$  and choose  $pad \in_R \{0, 1\}^{512}$
11. if ( $s \geq L$ )
12.    if ( $b = 0$ ) then goto step 5 else set  $w = s$
13. else
14.    if ( $b = 0$ ) then set  $w = max - s$  else set  $w = s$
15. let  $w_s$  be  $w$  represented as a binary string
16. set  $h_s = \text{SHA1}(m_s)$  and set  $\alpha_s = h_s \parallel m_s$
17. set  $u_s = pad \oplus \alpha_s$
18. set  $c_s = w_s \parallel u_s$  and set  $\tau = 1$
19. output ( $\tau, c_s$ )

The probability that  $z \notin \mathbb{Z}_n^*$  is negligible but we account for this possibility to ensure that  $s \in_R \mathbb{Z}_n$ . When  $z \notin \mathbb{Z}_n^*$  it follows that encryption has failed. We deliberately do not set  $\tau = 0$  in this case, since doing so might cause a user to not send the stegotext to the receiver and this would eliminate the uniformity of the stegotexts.

Steps 5 through 14 above are based on the probabilistic bias removal method [14]. The use of the PBRM ensures that  $w_s$  is uniformly distributed in  $\{0, 1\}^{|n_s|}$ .

YYPKSDecr( $c_s, n, d$ ):

Input: stegotext  $c_s$ , RSA modulus  $n$  and private exponent  $d$

Output: result code  $\tau \in \{0, 1\}$  and plaintext  $m_s \in \{0, 1\}^{352}$

1. set  $m_s = 0^{352}$  and set  $\tau = 0$
2. if (`IsAcceptableMod`( $n$ ) = 0) then
3.     print “bad modulus n” and goto step 19
4. let  $n_s$  be  $n$  represented as a binary string and set  $max = 2^{|n_s|}$
5. let  $w_s$  be the upper  $|n_s|$  bits of  $c_s$
6. let  $u_s$  be the lower 512 bits of  $c_s$
7. let  $w$  be the integer corresponding to  $w_s$
8. if ( $w \geq n$ ) then set  $s = max - w$  else set  $s = w$
9. if  $s \notin \mathbf{Z}_n^*$  then print “invalid RSA preimage” and goto step 19
10. compute  $r = s^d \bmod n$
11. let  $r_s$  be  $r$  represented as a binary string with leading zeros  
     added as needed to make  $|r_s| = |n_s|$  and set  $pad = \text{SHA512}(r_s)$
12. set  $\alpha_s = pad \oplus u_s$
13. let  $h_s$  be the upper 160 bits of  $\alpha_s$
14. let  $\beta_s$  be the lower 352 bits of  $\alpha_s$
15. if (`SHA1`( $\beta_s$ )  $\neq h_s$ ) then
16.     print “decryption failed”
17. else
18.     set  $(\tau, m_s) = (1, \beta_s)$
19. output  $(\tau, m_s)$

The algorithm `YYPKSHybridEncr` is used to hybrid encrypt plaintext. Observe that the plaintext  $m_s$  that is passed to `YYPKSEncr` is a uniformly random 352-bit string. This is deliberate even though some of the bits of  $m_s$  are not used.<sup>2</sup>

Define `CBCIDEAEncr`( $\mu_s, \ell_1, k, IV$ ) to be an algorithm that encrypts the byte stream  $\mu_s$  of length  $\ell_1$  bytes using the IDEA block cipher in cipher block chaining mode. The IDEA symmetric key is  $k$  and the initialization vector is  $IV$ . The output is the ciphertext that results.

Also, `CBCIDEAEncr` pads  $\mu_s$  as follows. It appends  $n \leq 8$  padding bytes of value  $n$  to the end of  $\mu_s$  in order to make the length of the resulting byte stream a multiple of the block size (8 in this case). If  $\mu_s$  is already a multiple of 8 bytes in length then 8 bytes of 0x08 are appended to the end of  $\mu_s$ . To illustrate, if 11 bytes are to be encrypted using IDEA then 5 padding bytes of value 0x05 will be appended. This is the padding method from PKCS #5 when the block size is 8 (see Section 6 of [12]).

Define `CBCIDEADecr`( $\theta_s, \ell_2, k, IV$ ) to be the corresponding CBC decryption algorithm. The CBC ciphertext is  $\theta_s$  and it is  $\ell_2$  bytes in length.

<sup>2</sup>Note that we are not saying that this is *necessary*, however.

Algorithm **CBCIDEADecr** returns  $\mu_s$  when  $\theta_s$  is a proper encryption of  $\mu_s$  under  $k$  and  $IV$ . Note that  $\ell_2 > \ell_1$  when  $\theta_s$  is a proper encryption of the  $\ell_1$ -byte plaintext  $\mu_s$ .

**YYPKSHybridEncr**( $\mu_s, n, e$ ):

Input: plaintext byte stream  $\mu_s \in \{0, 1\}^{8\ell_1}$ ,

RSA modulus  $n$  and corresponding RSA public exponent  $e$

Output: hybrid stegotext byte stream  $\theta_s$

1. choose  $m_s \in_R \{0, 1\}^{352}$
2. let  $k = m_s \bmod 2^{128}$  be an IDEA symmetric key
3. let  $IV = (m_s \gg 128) \bmod 2^{64}$  and  $\psi = ((m_s \gg 192) \bmod 8) + 1$
4. compute  $(\tau, c_s) = \text{YYPKSEncr}(m_s, n, e)$
5. if  $(\tau = 0)$  then print “error during encryption” and halt
6. choose  $r_s \in_R \{0, 1\}^{8\psi}$
7. compute  $\omega_s = \text{CBCIDEAEncr}(\mu_s, \ell_1, k, IV)$
8. set  $\theta_s = c_s \parallel r_s \parallel \omega_s$
9. output  $\theta_s$

The purpose of the random string  $r_s$  is to permit the size of the stegotext to vary. Encryption adds between 1 and 8 random bytes to the stegotext.

**YYPKSHybridDecr**( $\theta_s, n, d$ ):

Input: hybrid stegotext byte stream  $\theta_s$ ,

RSA modulus  $n$  and corresponding RSA private exponent  $d$

Output: plaintext byte stream  $\mu_s$

1. let  $n_s$  be  $n$  represented as a binary string
2. let  $c_s$  be the  $|n_s|$  most significant bits of  $\theta_s$
3. compute  $(\tau, m_s) = \text{YYPKSDecr}(c_s, n, d)$
4. if  $(\tau = 0)$  then print “error during decryption” and halt
5. let  $k = m_s \bmod 2^{128}$  be an IDEA symmetric key
6. let  $IV = (m_s \gg 128) \bmod 2^{64}$  and  $\psi = ((m_s \gg 192) \bmod 8) + 1$
7. solve for  $\ell_2$  in  $|\theta_s| = |n_s| + 8\psi + 8\ell_2$
8. let  $\omega_s$  be the  $\ell_2$  least significant bytes of  $\theta_s$
9. compute  $\mu_s = \text{CBCIDEADecr}(\omega_s, \ell_2, k, IV)$
10. output  $\mu_s$

In practice **CBCIDEADecr** is in a position to detect certain errors in the CBC ciphertext if present. This is due to the format of the padding that should exist in the final plaintext block.

## 4 Creating the Program

The program `yypks` consists of the ANSI C source file `yypks.c`. We constructed a simple makefile to create `yypks`. The program was compiled using `gcc`. We used the Minimalist GNU for Windows development suite (MinGW) and employed the OpenSSL crypto library for its underlying crypto routines. In particular we used OpenSSL version 0.9.8d. Our program also utilizes the Microsoft Cryptographic API (MS CAPI) to seed the OpenSSL random number generator. Compiling the program results in the MS DOS command-line program `yypks.exe`.

## 5 Running the Program

One of the first things that `yygen` does is seed the OpenSSL random number generator. It obtains 512 random bytes using the Microsoft Cryptographic API call `CryptGenRandom` and then passes them to the OpenSSL function `RAND_seed`. The program then calls `RAND_status` to check and see whether OpenSSL believes it has been seeded with enough random bytes.

The format for encryption is,

```
yypks e [cert file] [ptext file] [ctext file]
```

The `cert file` is an X.509 digital certificate [7, 8, 9] file that can either be in DER encoded binary format or base-64 encoded. A typical file extension for such a file is `.cer`. `ptext file` specifies the plaintext file that is encrypted and `ctext file` is the name of the resulting output ciphertext. The following is an example of how `yypks` is used to encrypt a plaintext file.

```
yypks e yytest.cer plaintext.txt ciphertext.bin
```

Decryption is accomplished using the command below. `PKCS #12 file` is the name of the input personal information exchange file [11]. Typical file extensions for this are `.p12` and `.pfx`. If this file contains an encrypted private key then the correct password must be specified in the `password` parameter. Parenthesis are required to enclose the password. If no password was used to secure the private key in the `PKCS #12 file` then the last argument should be `()`.

```
yypks d [PKCS #12 file] [ctext file] [ptext file] [(password)]
```

For example, if the PKCS #12 file `yyptest.p12` contains an RSA private key encrypted using the password “1234554321” then decryption of `ciphertext.bin` can be accomplished using,

```
yypks d yytest.p12 ciphertext.bin plaintext2.txt (1234554321)
```

## 6 Conclusion

An experimental public key stegosystem was presented that is based on RSA and the probabilistic bias removal method. The security of it was argued on heuristic grounds. The implementation of the experiment was described and examples were given on how to encrypt and decrypt files. We hope that this work will stimulate academic research in this area.

## References

- [1] M. Backes and C. Cachin. Public-key steganography with active attacks. In *Proceedings of the 2nd Theory of Cryptography Conference*, pages 210–226. Springer, 2005.
- [2] N. J. Hopper, J. Langford, and L. Von Ahn. Provably secure steganography. In Moti M. Yung, editor, *Advances in Cryptology—Crypto ’02*, pages 77–92. Springer, 2002. Lecture Notes in Computer Science No. 2442.
- [3] X. Lai and J. Massey. A proposal for a new block encryption standard. In I. B. Damgård, editor, *Advances in Cryptology—Eurocrypt ’90*, pages 389–404. Springer-Verlag, 1990. Lecture Notes in Computer Science No. 473.
- [4] X. Lai and J. Massey. Markov ciphers and differential cryptanalysis. In D. W. Davies, editor, *Advances in Cryptology—Eurocrypt ’91*, pages 17–38. Springer, 1991. Lecture Notes in Computer Science No. 547.
- [5] Bodo Möller. A public-key encryption scheme with pseudo-random ciphertexts. In *Proceedings of the 9th European Symposium on Research in Computer Security*, pages 335–351. Springer-Verlag, 2004.
- [6] National Institute of Standards and Technology (NIST). FIPS Publication 180-2: Announcing the Secure Hash Standard. August 1, 2002.

- 
- [7] ITU-T Rec. The Directory—Authentication Framework. International Telecommunication Union, Geneva, Switzerland, July 1993. X.509 (revised).
  - [8] ITU-T Rec. The Directory—Authentication Framework. International Telecommunication Union, Geneva, Switzerland, July 1995. X.509 (1988 and 1993) Technical Corrigendum 2.
  - [9] ITU-T Rec. The Directory—Authentication Framework. International Telecommunication Union, Geneva, Switzerland, July 1995. X.509 (1993) Amendment 1: Certificate Extensions.
  - [10] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.
  - [11] RSA Laboratories. *PKCS #12: Personal Information Exchange Syntax*, June 1999. Version 1.0.
  - [12] RSA Laboratories. *PKCS #5: Password-Based Cryptography Standard*, October 2006. Version 2.1.
  - [13] L. von Ahn and N. J. Hopper. Public-key steganography. In *Advances in Cryptology—Eurocrypt '04*, pages 323–341. Springer-Verlag, 2004.
  - [14] Adam L. Young and Moti M. Yung. Kleptography: Using cryptography against cryptography. In Walter Fumy, editor, *Advances in Cryptology—Eurocrypt '97*, pages 62–74. Springer-Verlag, 1997. Lecture Notes in Computer Science No. 1233.