

Errata for *Serious Cryptography* (updated to 8th printing)

Page 4: In the first paragraph, the sentence:

A cryptanalyst can then deduce that the key's length is either nine or a value **divisible by** nine (that is, three).

should now read:

A cryptanalyst can then deduce that the key's length is either nine or a value **that divides** nine (that is, three).

Page 13: The last sentence of the second paragraph under “Semantic Security and Randomized Encryption: IND-CPA” that reads:

Decryption remains deterministic, however, because given **E(K, R, P)**, you should always get *P*, regardless of the value of *R*.

should now read:

Decryption remains deterministic, however, because given **D(K, R, P)**, you should always get *P*, regardless of the value of *R*.

Page 14: In the second paragraph under “Achieving Semantically Secure Encryption,” all instances of “**DRBG(K, R)**” should now read “**DRBG(K || R)**”

Page 28: The equation that shows:

$$S_{k+624} = S_{k+397} \oplus A((S_k \wedge 0x80000000) \vee (S_{k+1} \wedge 0xffffffff))$$

should now show:

$$S_{k+624} = S_{k+397} \oplus A((S_k \wedge 0x80000000) \vee (S_{k+1} \wedge 0x7fffffff))$$

Page 32: The caption for Listing 2-3 that reads:

A script showing the evolution of `/dev/urandom`'s entropy estimate
should now read:

A script showing the evolution of `/dev/random`'s entropy estimate

Page 49: The terminal command that reads:

`openssl rand 16 -hex`

should now read:

`openssl rand -hex 16`

Page 70: In the second paragraph under “Ciphertext Stealing,” the sentence that reads:

The last, incomplete ciphertext block is made up of the first **blocks** from the previous ciphertext block . . .

should now read:

The last, incomplete ciphertext block is made up of the first **bits** from the previous ciphertext block . . .

Page 73: The equation in the second paragraph that reads:

$E(K_1, E(K_2, P))$

should now read:

$E(K_2, E(K_1, P))$

and in the last paragraph, the sentence:

You also need to store 2^{56} elements of 15 bytes each, or about **128 petabytes**.

should now read:

You also need to store 2^{56} elements of 15 bytes each, or about **1 exabyte**.

Page 74: In Figure 4-13, the two boxes that are labeled **E_k** should now be labeled **D_k**

and in the fourth paragraph, the part of the sentence that reads:

. . . decryption will only succeed if $C_1 \oplus P_2 = X$ ends with valid padding . . .

should now read:

. . . decryption will only succeed if $C_1 \oplus X = P_2$ ends with valid padding . . .

Page 84: The state values after “Repeating the operation four times . . .” that show:

0 1 1 1
1 1 0 0
1 0 0 0
0 0 0 1

should instead show:

0 1 1 1
1 1 1 0
1 1 0 0
1 0 0 0

and the subsequent paragraph that reads:

And as you can see, the state after **five** updates is the same as the initial one, demonstrating that we're in a **period-5** cycle and proving that the LFSR's period isn't the maximal value of 15.
should now read:

And as you can see, the state after **six** updates is the same as the initial one, demonstrating that we're in a **period-6** cycle and proving that the LFSR's period isn't the maximal value of 15.

Page 92: In the first paragraph of the RC4 section, “**Wireless** Equivalent Privacy” should now read “**Wired** Equivalent Privacy.” The acronym list should also reflect this change.

Page 100: In the third paragraph, the sentence that reads:

The brute-forcing takes 2^{36} operations, a computation **that dwarfs** the unrealistic $2^{220} * 2^{31} = 2^{251}$ trials . . .

should now read:

The brute-forcing takes 2^{36} operations, a computation **that is dwarfed by** the unrealistic $2^{220} * 2^{31} = 2^{251}$ trials . . .

Page 107: The SHA-256 hash values for **a**, **b**, and **c** which read:

SHA-256 ("a") =
87428fc522803d31065e7bce3cf03fe475096631e5e07bbd7a0fde60c4cf25c7
SHA-256 ("b") =
a63d8014dba891345b30174df2b2a57efbb65b4f9f09b98f245d1b3192277ece

```
SHA-256 ("c") =  
edeaff3f1774ad2888673770c6d64097e391bc362d7d6fb34982ddf0efd18cb
```

should now read:

```
SHA-256 ("a") =  
ca978112calbbdcfac231b39a23dc4da786eff8147c4e72b9807785afee48bb  
SHA-256 ("b") =  
3e23e8160039594a33894f6564e1b1348bbd7a0088d42c4acb73eeaed59c009d  
SHA-256 ("c") =  
2e7d2c03a9507ae265ecf5b5356885a53393a2029d241394997265a1a25aefc6
```

Page 108: Listing 6-2 which reads:

```
solve-second-preimage(M) {  
    H = Hash(M)  
    return solve-preimage(H)  
}
```

should now read:

```
find-second-preimage(M) {  
    H = Hash(M)  
    return find-preimage(H)  
}
```

and `solve-preimage` should now read `find-preimage` in the sentence preceding the listing.

We also updated the name of this section from “Why Second-Preimage Resistance Is Weaker” to “From Preimages to Second Preimages.”

Page 130: Under “Creating Keyed Hashes from Unkeyed Hashes,” the part of the sentence:

... usually hash functions **of** block ciphers.

should now read:

... usually hash functions **or** block ciphers.

Page 141: In Listing 7-2, # each call to `verify_mac()` will look at all **eight** bytes
should now read # each call to `verify_mac()` will look at all **sixteen** bytes

Page 147: In the paragraph under “Encrypt-then-MAC,” the sentence which reads:

If the values are equal, the plaintext is computed as $\mathbf{P} = D(K_1, C)$; if they are not equal, the **plaintext** is discarded.

should now read:

If the values are equal, the plaintext is computed as $\mathbf{P} = D(K_1, C)$; if they are not equal, the **ciphertext** is discarded.

Page 152: The sentence beginning:

To authenticate the ciphertext, GCM uses a Wegman–Carter MAC (see Chapter 7) **to authenticate the ciphertext**, which XORs the value . . .

should now read:

To authenticate the ciphertext, GCM uses a Wegman–Carter MAC (see Chapter 7) which XORs the value . . .

Page 153: “GHASH(**H, C**)” should now read “GHASH(**H, A, C**)”

Page 154: The equation for T_2 which reads:

$$T_2 = \mathbf{GHASH}(H, A_1, C_1) + \mathbf{AES}(K, N \parallel 0)$$

should now read:

$$T_2 = \mathbf{GHASH}(H, A_2, C_2) + \mathbf{AES}(K, N \parallel 0)$$

Page 165: In the fourth paragraph, the part of the sentence that reads:

. . . when we say that an algorithm takes time in the order of n^3 operations (which is **quadratic complexity**), . . .

should now read:

. . . when we say that an algorithm takes time in the order of n^3 operations (which is **cubic complexity**), . . .

Page 175: The final sentence before section “The Hard Thing” which reads:

Typically, we’ll use groups Z_p^* , where p is *thousands* of bits long (that is, groups that contain on the order of 2^p numbers).

should now read:

Typically, we'll use groups Z_p^* , where p is *thousands* of bits long (that is, groups that contain on the order of 2^m numbers if p is m -bit long).

Page 181: The last sentence of the first paragraph which reads:

(One year prior to RSA, Diffie and Hellman had introduced the concept of public-key cryptography, but their scheme was unable to perform public-key encryption.)
should now read:

(One year prior to RSA, Diffie and Hellman had introduced the concept of public-key cryptography, but their scheme was unable to perform public-key signatures.)

Page 182: In the second paragraph under "The Math Behind RSA," the sentences that read:

More precisely, RSA works on the numbers less than n that are co-prime with n and therefore that have no common prime factor with n . Such numbers, when multiplied together, yield another number that satisfies these criteria. We say that these numbers form a group, denoted Z_n^* , and call the multiplicative group of integers modulo n .

should now read:

Such numbers, when multiplied together, yield another number that satisfies these criteria. We say that these numbers form a group, denoted Z_n^* , and call it the multiplicative group of integers modulo n .

Page 183: We deleted the last sentence of the second paragraph beginning "In other words . . ."

and " $ed = 1 \bmod \phi(n)$ " should now read " $ed \bmod \phi(n) = 1$ "

Page 184: The part of Listing 10-1 that reads:

```
sage: n = p*q; n  
c  
sage: phi = (p-1)*(q-1); phi  
36567230045260644  
sage: e = random_prime(phi); e  
13771927877214701
```

```
sage: d = xgcd(e, phi)[1]; d  
15417970063428857
```

should now read:

```
sage: n = p*q; n  
19715247602230861  
sage: phi = (p-1)*(q-1); phi  
19715246481137724  
sage: e = random_prime(phi); e  
13771927877214701  
sage: d = e.inverse_mod(phi); d  
11417851791646385
```

and in the description under the listing, the part that reads:

We then generate the associated private exponent d by using the `xgcd()` function from Sage (6).

should now read:

We then generate the associated private exponent d by using the `inverse_mod()` function from Sage (6).

Page 189: In the fourth paragraph under “The PSS Signature Standard,” “signature” should read “signatures”

and in the second paragraph, the sentence that reads:

Here’s how this works: because S can be written as $(R^eM)^d = R^{ed}M^d$, and because $R^{ed} = R$ is equal to $R^{ed} = R$ (by definition) . . .

should now read:

Here’s how this works: because S can be written as $(R^eM)^d = R^{ed}M^d$, and because $R^{ed} = R$ (by definition) . . .

Page 191: In the third paragraph under “RSA Implementations,” the sentence that reads:

The function `EncryptoAEP()` takes a hash `value`, a PRNG, a public key, a message, and a label (an optional parameter of OAEP), and returns a signature and an error code.

should now read:

The function `EncryptOAEP()` takes a hash **function**, a PRNG, a public key, a message, and a label (an optional parameter of OAEPE), and returns a ciphertext and an error code.

Page 193: In Listing 10-5, `for i = m - 1` should now read `for i = m - 2`

Page 195: In the first paragraph under “The Chinese Remainder Theorem,” the sentence:

The most common trick to speed up decryption and signature **verification** (that is, the computation of $y^d \bmod n$) is the Chinese remainder theorem (CRT).

should now read:

The most common trick to speed up decryption and signature **generation** (that is, the computation of $y^d \bmod n$) is the Chinese remainder theorem (CRT).

Page 196: In the first paragraph, the sentence that reads:

To apply this formula to our example and recover our $x \bmod 1155$, we take the arbitrary values 2, 1, 6, and 8; we compute $P(3)$, $P(5)$, $P(7)$, and $P(8)$; and then we add them together to get the following expression:

should now read:

To apply this formula to our example and recover our $x \bmod 1155$, we take the arbitrary values 2, 1, 6, and 8; we compute $P(3)$, $P(5)$, $P(7)$, and $P(11)$; and then we add them together to get the following expression:

Page 198: φ should be superscript in the sentence:

... we know that for any number a co-prime with n , $a^{\varphi(n)} = 1 \bmod n$...

and in the sentence:

That is, we'll be able to write $a^k \varphi(n) = 1 \bmod n$...

and the code line labeled with number ball (5) that reads:

```
if x != 1 and x != (n - 1) and pow(x, 2, n) == 1:
```

should now read:

```
if x != 1 and x != (n - 1) and pow(x, 2, n) == 1:
```

Page 213: In the bottom-right formula for Bob, “ $(A \times Y^A)$ ” should now read “ $(A \times X^A)$ ”

Page 215: In the first paragraph, the equation “ $g^4 = 13$ ” should now read “ $g^4 = 3$ ”

and the sentence that reads:

The TLS protocol is the security behind HTTPS secure websites as well as the **secure mail transfer protocol** (SMTP).

should now read:

The TLS protocol is the security behind HTTPS secure websites as well as the **Simple Mail Transfer Protocol** (SMTP).

Page 220: The sentence in the Note which reads:

Such a set of numbers, where addition and multiplication are possible and where each element x admits an inverse with respect to addition (denoted $-x$) as well as an inverse with respect to multiplication (denoted $1/x$), is called a *field*.

should now read:

Such a set of numbers, where addition and multiplication are possible and where each element x admits an inverse with respect to addition (denoted $-x$) as well as an **inverse (except for the element zero)** with respect to multiplication (denoted $1/x$), is called a *field*.

Page 221: The second sentence in the first paragraph under “Adding Two Points” that reads:

... and Q is the reflection of this point with respect to the x-axis.

should now read:

... and R is the reflection of this point with respect to the x-axis.

Page 222: The first sentence on the page which reads:

... and if $P = -P$, ...

should now read:

... and if $Q = -P$, ...

Page 225: In the last set of equations on the page,

$$k(d_1 - d_2) = c_1 - c_2$$

and

$$k = (c_1 - c_2) / (\textcolor{red}{d}_1 - \textcolor{red}{d}_2)$$

should now read:

$$k(\textcolor{red}{d}_2 - \textcolor{red}{d}_1) = c_1 - c_2$$

and

$$k = (c_1 - c_2) / (\textcolor{red}{d}_2 - \textcolor{red}{d}_1)$$

Page 227: The second equation that reads:

$$wr = rk(h + rd) = v$$

should now read:

$$wr = rk / (h + rd) = v$$

and the fifth equation on that reads:

$$u + vd = hk(h + rd) + drk(h + rd) = (hk + drk)(h + rd) = k(h + dr)(h + rd) = k$$

should now read:

$$u + vd = hk / (h + rd) + drk / (h + rd) = (hk + drk) / (h + rd) = k(h + dr) / (h + rd) = k$$

Page 228: The first sentence of the second paragraph which reads:

RSA's verification process is often faster than ECC's signature **generation** because it uses a small public key e .

should now read:

RSA's verification process is often faster than ECC's signature **verification** because it uses a small public key e .

Page 229: The last sentence before the section "Choosing a Curve" which reads:

Decryption is straightforward: the recipient computes S by multiplying $\textcolor{red}{R}$ with their private exponent to obtain S , and then derives the key K and decrypts C and verifies T .

should now read:

Decryption is straightforward: the recipient computes S by multiplying $\textcolor{red}{Q}$ with their private exponent to obtain S , and then derives the key K and decrypts C and verifies T .

Page 239: The last sentence in the first paragraph that reads:

The organization that issued certificate [2](#) (GeoTrust) granted permission to Google Internet Authority to issue a certificate (certificate [1](#)) for the domain name www.google.com, thereby transferring trust to Google Internet Authority.
should now read:

The organization that issued certificate [1](#) (GeoTrust) granted permission to Google Internet Authority to issue a certificate (certificate [0](#)) for the domain name www.google.com, thereby transferring trust to Google Internet Authority.

Page 242: We deleted the paragraph starting with “But note that the specifications. . .” because the content is repeated from the first paragraph of the section.

Page 243: We deleted the paragraph starting with “Note, however, that TLS 1.3 supports many options and extensions . . .” because the information is repeated in the note below.

Page 254: The first equation which shows:

$$\Psi = \left(1 / \sqrt{2}\right)|0\rangle + \left(1 / \sqrt{2}\right)|1\rangle = (|0\rangle + |1\rangle) / \sqrt{2}$$

should now show:

$$\Psi = \left(1 / \sqrt{2}\right)|0\rangle + \left(1 / \sqrt{2}\right)|1\rangle = (|0\rangle + |1\rangle) / \sqrt{2}$$

and the second equation which shows:

$$\Phi = \left(i / \sqrt{2}\right)|0\rangle - \left(1 / \sqrt{2}\right)|1\rangle = (i|0\rangle - |1\rangle) / \sqrt{2}, \text{ or } |\Phi\rangle = \left(i / \sqrt{2}, 1 / \sqrt{2}\right)$$

should now show:

$$\Phi = \left(i / \sqrt{2}\right)|0\rangle - \left(1 / \sqrt{2}\right)|1\rangle = (i|0\rangle - |1\rangle) / \sqrt{2}, \text{ or } |\Phi\rangle = \left(i / \sqrt{2}, -1 / \sqrt{2}\right)$$

Page 257: The first equation which shows:

$$H|\Psi\rangle = \begin{pmatrix} 1 / \sqrt{2} & 1 / \sqrt{2} \\ 1 / \sqrt{2} & -1 / \sqrt{2} \end{pmatrix} \begin{pmatrix} 1 / \sqrt{2} \\ 1 / \sqrt{2} \end{pmatrix} = \begin{pmatrix} 1 / 2 + 1 / 2 \\ 1 / 2 - 1 / 2 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} = 0$$

should now show:

$$H|\Psi\rangle = \begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{pmatrix} \begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix} = \begin{pmatrix} 1/2 + 1/2 \\ 1/2 - 1/2 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} = |0\rangle$$

Page 260: The first paragraph under “Shor’s Algorithm and the Discrete Logarithm Problem”: The challenge in the discrete logarithm problem is to find y , given $y = g^x \bmod p$, for some known numbers g and p . Solving this problem takes an exponential amount of time on a classical computer, but Shor’s algorithm lets you find y easily thanks to its efficient period-finding technique.

should now read:

The challenge in the discrete logarithm problem is to find x , given $y = g^x \bmod p$, for some known numbers g and p . Solving this problem takes an exponential amount of time on a classical computer, but Shor’s algorithm lets you find x easily thanks to its efficient period-finding technique.

Page 264: The arrow placement for Figure 14-5 is slightly inaccurate. Please refer to the below figure instead:

