

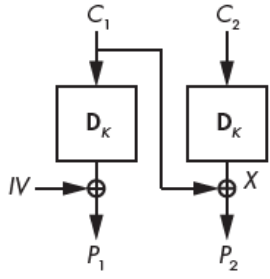
Serious Cryptography

A Practical Introduction to Modern Encryption

by Jean-Philippe Aumasson

errata updated to print 9

Page	Error	Correction	Print corrected
4	A cryptanalyst can then deduce that the key's length is either nine or a value divisible by nine (that is, three)	A cryptanalyst can then deduce that the key's length is either nine or a value that divides nine (that is, three)	Print 2
13	Decryption remains deterministic, however, because given $E(K, R, P)$, you should always get P , regardless of the value of R .	Decryption remains deterministic, however, because given $D(K, R, P)$, you should always get P , regardless of the value of R .	Print 2
14	The proof works ad absurdum: if you can distinguish ciphertexts from random strings, which means that you can distinguish $DRBG(K, R) \oplus P$ from random, then this means that you can distinguish $DRBG(K, R)$ from random. Remember that the CPA model lets you get ciphertexts for chosen values of P , so you can XOR P to $DRBG(K, R) \oplus P$ and get $DRBG(K, R)$. But now we have a contradiction, because we started by assuming that $DRBG(K, R)$ can't be distinguished from random, producing random strings.	The proof works ad absurdum: if you can distinguish ciphertexts from random strings, which means that you can distinguish $DRBG(K R) \oplus P$ from random, then this means that you can distinguish $DRBG(K R)$ from random. Remember that the CPA model lets you get ciphertexts for chosen values of P , so you can XOR P to $DRBG(K R) \oplus P$ and get $DRBG(K R)$. But now we have a contradiction, because we started by assuming that $DRBG(K R)$ can't be distinguished from random, producing random strings.	Print 2
16	<i>Authenticated encryption with associated data (AEAD)</i> is an extension of authenticated encryption that takes some cleartext and unencrypted data and uses it to generate the authentication tag $AEAD(K, P, A) = (C, T)$.	<i>Authenticated encryption with associated data (AEAD)</i> is an extension of authenticated encryption that takes some cleartext and unencrypted data and uses it to generate the authentication tag $AEAD(K, P, A) = (C, A, T)$.	Print 9
28	Equation replacement	$S_{k+624} = S_{k+397} \oplus A((S_k \wedge 0x80000000) \vee (S_{k+1} \wedge 0x7fffffff))$	Print 2
32	<i>Listing 2-3: A script showing the evolution of /dev/urandom's entropy estimate</i>	<i>Listing 2-3: A script showing the evolution of /dev/random's entropy estimate</i>	Print 2
46	We say that breaking some cipher is reducible to problem X if any method to solve problem X also yields a method to break the cipher .	We say that solving problem X is reducible to breaking some cipher if any method to break the cipher can be efficiently adapted to solve problem X .	Print 9
49	<pre>openssl rand 16 -hex</pre>	<pre>openssl rand -hex 16</pre>	Print 6
70	The last, incomplete ciphertext block is made up of the first blocks from the previous ciphertext block . . .	The last, incomplete ciphertext block is made up of the first bits from the previous ciphertext block . . .	Print 2
73	Why use triple DES and not just double DES, that is, $E(K_1, E(K_2, P))$?	Why use triple DES and not just double DES, that is, $E(K_2, E(K_1, P))$?	Print 2
73	You also need to store 2^{56} elements of 15 bytes each, or about 128 petabytes .	You also need to store 2^{56} elements of 15 bytes each, or about 1 exabyte .	Print 2

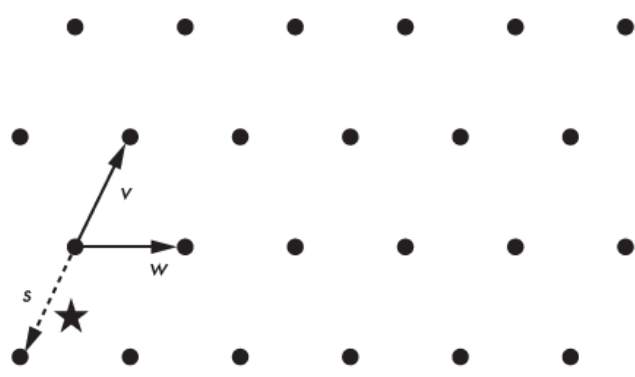
Page	Error	Correction	Print corrected
74	Figure replacement	 <p data-bbox="1066 495 1333 641"><i>Figure 4-13: Padding oracle attacks recover X by choosing C₁ and checking the validity of padding.</i></p>	Print 6
74	... decryption will only succeed if $C_1 \oplus P_2 = X$ ends with valid padding decryption will only succeed if $C_1 \oplus X = P_2$ ends with valid padding ...	Print 6
84	Repeating the operation four times gives the following state values: 0 1 1 1 1 1 0 0 1 0 0 0 0 0 0 1 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.	Repeating the operation four times gives the following state values: 0 1 1 1 1 1 1 0 1 1 0 0 1 0 0 0 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.	Print 6
92	Wireless Equivalent Privacy	Wired Equivalent Privacy	Print 2
100	The brute-forcing takes 2^{36} operations, a computation that dwarfs the unrealistic $2^{220} \times 2^{31} = 2^{251}$ trials needed to find the 220 bits to complete the first part of the attack.	The brute-forcing takes 2^{36} operations, a computation that is dwarfed by the unrealistic $2^{220} \times 2^{31} = 2^{251}$ trials needed to find the 220 bits to complete the first part of the attack.	Print 2
107	<pre data-bbox="178 1193 1018 1388"> SHA-256("a") = 87428fc522803d31065e7bce3cf03fe475096631e5e07bbd7a0fde60c4cf25c7 SHA-256("b") = a63d8014dba891345b30174df2b2a57efbb65b4f9f09b98f245d1b3192277ece SHA-256("c") = edeaaaf3f1774ad2888673770c6d64097e391bc362d7d6fb34982ddf0efd18cb </pre>	<pre data-bbox="1050 1193 1879 1388"> SHA-256("a") = ca978112ca1bbdcafacc231b39a23dc4da786eff8147c4e72b9807785afee48bb SHA-256("b") = 3e23e8160039594a33894f6564e1b1348bbd7a0088d42c4acb73eeaed59c009d SHA-256("c") = 2e7d2c03a9507ae265ecf5b5356885a53393a2029d241394997265a1a25aefc6 </pre>	Print 2

Page	Error	Correction	Print corrected
108	<p>As proof, if the algorithm solve-preimage() returns a preimage of a given hash value, you can use the algorithm in Listing 6-2 to find a second preimage of some message, <i>M</i>.</p> <pre>solve-second-preimage(M) { H = Hash(M) return solve-preimage(H) }</pre>	<p>As proof, if the algorithm find-preimage() returns a preimage of a given hash value, you can use the algorithm in Listing 6-2 to find a second preimage of some message, <i>M</i>.</p> <pre>find-second-preimage(M) { H = Hash(M) return find-preimage(H) }</pre>	Print 6
117	<pre>for i = 0 to 79 { new = (a <<< 5) + f(i, b, c, d) + e + K[i] + W[i]</pre>	<pre>for i = 0 to 79 { new = (a <<< 5) + f(i, b, c, d) + e + M[i] + W[i]</pre>	Print 9
130	Throughout the history of cryptography, MACs and PRFs have rarely been designed from scratch but rather have been built from existing algorithms, usually hash functions of block ciphers.	Throughout the history of cryptography, MACs and PRFs have rarely been designed from scratch but rather have been built from existing algorithms, usually hash functions or block ciphers.	Print 6
141	<pre># each call to verify_mac() will look at all eight bytes</pre>	<pre># each call to verify_mac() will look at all sixteen bytes</pre>	Print 2
147	If the values are equal, the plaintext is computed as $P = \mathbf{D}(K_1, C)$; if they are not equal, the plaintext is discarded.	If the values are equal, the plaintext is computed as $P = \mathbf{D}(K_1, C)$; if they are not equal, the ciphertext is discarded.	Print 2
152	To authenticate the ciphertext, GCM uses a Wegman–Carter MAC (see Chapter 7) to authenticate the ciphertext , which XORs the value . . .	To authenticate the ciphertext, GCM uses a Wegman–Carter MAC (see Chapter 7) which XORs the value . . .	Print 2
153	We can thus express the authentication tag's value as $T = \mathbf{GHASH}(H, C)$	We can thus express the authentication tag's value as $T = \mathbf{GHASH}(H, A, C)$	Print 8
165	. . . when we say that an algorithm takes time in the order of n^3 operations (which is quadratic complexity), when we say that an algorithm takes time in the order of n^3 operations (which is cubic complexity), . . .	Print 2
172	$1/\log\sqrt{N} = 1/(n/2) = 2n$	$1/\log\sqrt{N} = 1/(n/2) = 2/n$	Print 9
175	Typically, we'll use groups \mathbf{Z}_p^* , where p is <i>thousands</i> of bits long (that is, groups that contain on the order of 2^p numbers).	Typically, we'll use groups \mathbf{Z}_p^* , where p is <i>thousands</i> of bits long (that is, groups that contain on the order of 2^m numbers if p is m-bit long).	Print 3
181	(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 .)	(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 .)	Print 2
182	Deletion	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 \mathbf{Z}_N^* , and call the multiplicative group of integers modulo n .	Print 2

Page	Error	Correction	Print corrected
183	Deletion	In other words, all but $(p + q)$ numbers between 1 and $n - 1$ belong to \mathbb{Z}_n^* and are “valid numbers” in RSA operations.	Print 2
183	More precisely, we must have $ed = 1 \bmod \phi(n)$ in order to get $x^{ed} = x^1 = x$ and to decrypt the message correctly.	More precisely, we must have $ed \bmod \phi(n) = 1$ in order to get $x^{ed} = x^1 = x$ and to decrypt the message correctly.	Print 8
184	<pre>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</pre>	<pre>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</pre>	Print 6
184	We then generate the associated private exponent d by using the <code>xgcd()</code> function from Sage ⑥	We then generate the associated private exponent d by using the <code>inverse_mod()</code> function from Sage ⑥	Print 6
184	Insertion	We then generate a random public exponent, e ⑥, by picking a random prime less than ϕ in order to ensure that e will have an inverse modulo ϕ . Note that e should also be coprime with ϕ.	Print 6
189	Here’s how this works: because S can be written as $(R^e M)^d = R^{ed} M^d$, and because $R^{ed} = R$ is equal to $R^{ed} = R$ (by definition) . . .	Here’s how this works: because S can be written as $(R^e M)^d = R^{ed} M^d$, and because $R^{ed} = R$ (by definition) . . .	Print 2
189	Although similar to PSS, OAEP has only been proven secure for encryption, not for signature .	Although similar to PSS, OAEP has only been proven secure for encryption, not for signatures .	Print 2
191	The function <code>EncryptOAEP()</code> 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.	The function <code>EncryptOAEP()</code> takes a hash function, a PRNG, a public key, a message, and a label (an optional parameter of OAEP), and returns a ciphertext and an error code.	Print 6
193	<pre>expMod(x, e, n) { y = x for i = n - 1 to 0 {</pre>	<pre>expMod(x, e, n) { y = x for i = n - 2 to 0 {</pre>	Print 6
195	The most common trick to speed up decryption and signature verification (that is, the computation of $y^d \bmod n$) is the <i>Chinese remainder theorem (CRT)</i> .	The most common trick to speed up decryption and signature generation (that is, the computation of $y^d \bmod n$) is the <i>Chinese remainder theorem (CRT)</i> .	Print 2
196	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:	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:	Print 3
198	. . . we know that for any number a co-prime with n , $a^{\phi(n)} = 1 \bmod n$ we know that for any number a co-prime with n , $a^{\phi(n)} = 1 \bmod n$.	Print 3

Page	Error	Correction	Print corrected
198	That is, we'll be able to write $a^{k\phi(n)} = 1 \pmod n \dots$	That is, we'll be able to write $a^{k\phi(n)} = 1 \pmod n \dots$	Print 3
198	<pre>⑤ if x != 1 and x != (n - 1) and pow(x, 2, n) == 1:</pre>	<pre>⑤ if x != 1 and x != (n - 1) and pow(x, 2, n) == 1:</pre>	Print 3
213	Figure replacement	<p>Figure 11-6: The MQV protocol</p>	Print 8
215	$\dots g^1 = 2, g^2 = 4, g^3 = 8, g^4 = 13$, and so on.	$\dots g^1 = 2, g^2 = 4, g^3 = 8, g^4 = 3$, and so on.	Print 2
215	The TLS protocol is the security behind HTTPS secure websites as well as the secure mail transfer protocol (SMTP).	The TLS protocol is the security behind HTTPS secure websites as well as the Simple Mail Transfer Protocol (SMTP).	Print 4
220	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.	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.	Print 3
221	\dots and Q is the reflection of this point with respect to the x-axis.	\dots and R is the reflection of this point with respect to the x-axis.	Print 2
222	\dots and if $P = -P, \dots$	\dots and if $Q = -P, \dots$	Print 3
225	Equation replacement	$d_2k - d_1k = c_1 - c_2$ $k(d_2 - d_1) = c_1 - c_2$ $k = (c_1 - c_2) / (d_2 - d_1)$	Print 2
227	$wr = rk(b + rd) = v$	$wr = rk / (b + rd) = v$	Print 2
227	$u + vd = bk(b + rd) + drk(b + rd) = (bk + drk)(b + rd) = k(b + dr)(b + rd) = k$	$u + vd = bk / (b + rd) + drk / (b + rd) = (bk + drk) / (b + rd) = k(b + dr) / (b + rd) = k$	Print 2
228	RSA's verification process is often faster than ECC's signature generation because it uses a small public key e .	RSA's verification process is often faster than ECC's signature verification because it uses a small public key e .	Print 3

Page	Error	Correction	Print corrected
229	Decryption is straightforward: the recipient computes S by multiplying R with their private exponent to obtain S , and then derives the key K and decrypts C and verifies T .	Decryption is straightforward: the recipient computes S by multiplying Q with their private exponent to obtain S , and then derives the key K and decrypts C and verifies T .	Print 3
239	The organization that issued certificate 2 (GeoTrust) granted permission to Google Internet Authority to issue a certificate (certificate 1) for the domain name <i>www.google.com</i> , thereby transferring trust to Google Internet Authority.	The organization that issued certificate 1 (GeoTrust) granted permission to Google Internet Authority to issue a certificate (certificate 0) for the domain name <i>www.google.com</i> , thereby transferring trust to Google Internet Authority.	Print 2
242	Deletion	But note that the specifications also describe in what format data should be sent, in order to ensure interoperability between implementations by guaranteeing that any server implementing TLS 1.3 will be able to read TLS 1.3 data sent by any client implementing TLS 1.3, possibly using a different library or programming language.	Print 6
243	Deletion	Note, however, that TLS 1.3 supports many options and extensions, so it may behave differently than what has been described here (and shown in Figure 13-1). You can, for example, configure the TLS 1.3 handshake to require a client certificate so that the server verifies the identity of the client. TLS 1.3 also supports a handshake with pre-shared keys.	Print 2
254	Equation replacement	$\Psi = (1/\sqrt{2}) 0\rangle + (1/\sqrt{2}) 1\rangle = (0\rangle + 1\rangle) / \sqrt{2}$	Print 3
254	Equation replacement	$\Phi = (i/\sqrt{2}) 0\rangle - (1/\sqrt{2}) 1\rangle = (i 0\rangle - 1\rangle) / \sqrt{2}, \text{ or } \Phi\rangle = (i/\sqrt{2}, -1/\sqrt{2})$	Print 3
257	Equation replacement	$H \Psi\rangle = \begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{pmatrix} \begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix} = \begin{pmatrix} 1/2 + 1/2 \\ 1/2 - 1/2 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} = 0\rangle$	Print 3
260	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.	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.	Print 2

Page	Error	Correction	Print corrected
264	Figure replacement	 <p data-bbox="1050 560 1680 657"><i>Figure 14-5: Points of a two-dimensional lattice, where v and w are basis vectors of the lattice, and s is the closest vector to the star-shaped point</i></p>	Print 2