D

DATA ENCRYPTION STANDARD (DES)

The Data Encryption Standard (DES) [31] has been around for more than 25 years. During this time the standard was revised three times: as FIPS-46-1 in 1988, as FIPS-46-2 in 1993 and as FIPS-46-3 in 1999. DES was an outcome of a call for primitives in 1974, which did not result in many serious candidates except for a predecessor of DES, Lucifer [15, 36] designed by IBM around 1971. It took another year for a joint IBM–NSA effort to turn *Lucifer* into DES. The structure of Lucifer was significantly altered: since the design rationale was never made public and the secret key size was reduced from 128-bit to 56-bits, this initially resulted in controversy, and some distrust among the public. After some delay, FIPS-46 was published by NBS (National Bureau of Standards)—now NIST (National Institute of Standards and Technology)—on January 15, 1977 [31] (see [35] for a discussion of the standardization process).

However, in spite of all the controversy it is hard to underestimate the role of DES [31]. DES was one of the first commercially developed (as opposed to government developed) ciphers whose structure was fully published. This effectively created a community of researchers who could analyse it and propose their own designs. This lead to a wave of public interest in cryptography, from which much of the cryptography as we know it today was born.

DESCRIPTION OF DES: The Data Encryption Standard, as specified in FIPS Publication 46-3 [31], is a block cipher operating on 64-bit data blocks. The encryption transformation depends on a 56-bit secret key and consists of sixteen Feistel iterations surrounded by two permutation layers: an initial bit permutation IP at the input, and its inverse IP^{-1} at the output. The structure of the cipher is depicted in Figure 1. The decryption process is the same as the encryption, except for the order of the round keys used in the Feistel iterations. As a result, most of the circuitry can be reused in hardware implementations of DES.

The 16-round Feistel network, which constitutes the cryptographic core of DES, splits the 64-bit data blocks into two 32-bit words (denoted by L_0 and R_0). In each iteration (or round), the second word R_i is fed to a function f and the result is added to the first word L_i. Then both words are swapped and the algorithm proceeds to the next iteration.

The function f is key-dependent and consists of four stages (see Figure 2). Their description is given below. Note that all bits in DES are numbered from left to right, i.e., the leftmost bit of a block (the most significant bit) is bit 1.

1. **Expansion (*E*).** The 32-bit input word is first expanded to 48 bits by duplicating and reordering half of the bits. The selection of bits is specified by Table 1. The first row in the table refers to the first 6 bits of the expanded word, the second row to bits 7–12, and so on. Thus bit 41 of the expanded word, for example, gets its value from bit 28 of the input word.
2. **Key mixing.** The expanded word is XORed with a round key constructed by selecting 48 bits from the 56-bit secret key. As explained below, a different selection is used in each round.

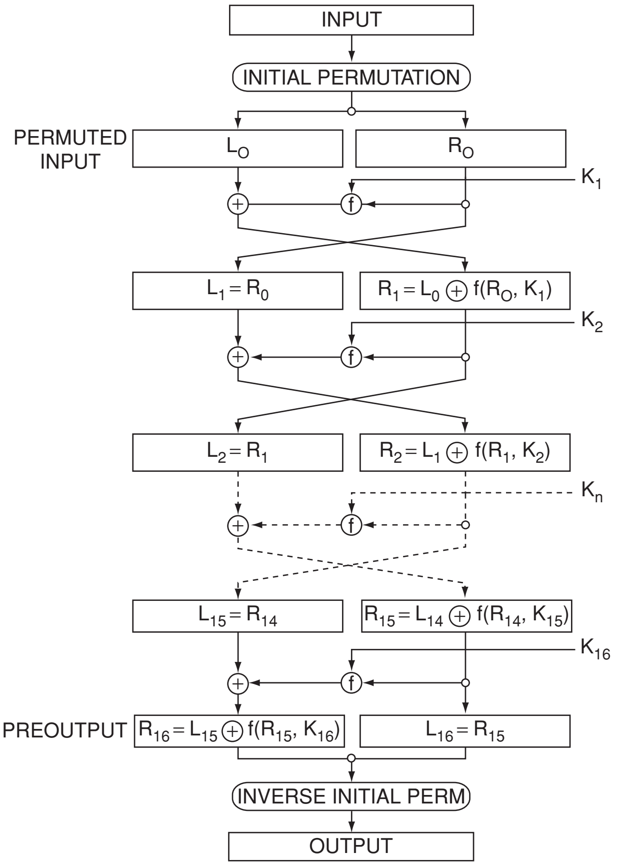

Fig. 1. The encryption function

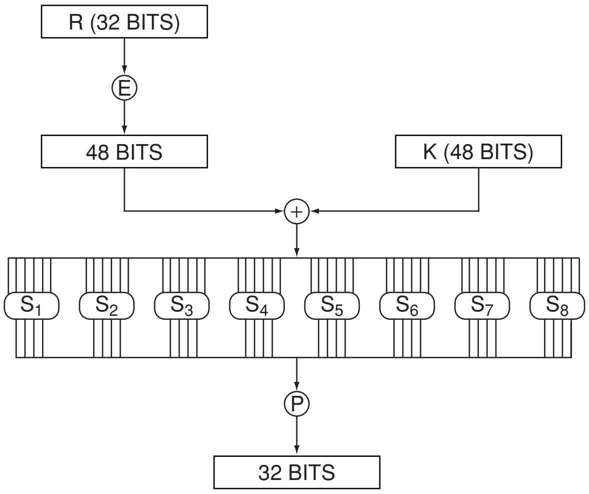

Fig. 2. The function f

3. **Substitution.** The 48-bit result is split into eight 6-bit words which are substituted in eight parallel 6×4-bit S-boxes. All eight S-boxes, called $S_1, S_2, \ldots, S_8$, are different but have the same special structure, as appears from their specifications in Table 2. Each row of the S-box tables consists of a permutation of the 4-bit values $0, \ldots, 15$. The 6-bit input word is substituted as follows: first a row is selected according to the value of the binary word formed by concatenating the first and the sixth input bit. The algorithm then picks the column given by the value of the four middle bits and outputs the corresponding 4-bit word.
4. **Permutation (P).** The resulting 32 bits are reordered according to a fixed permutation specified in Table 1 before being sent to the output. As before, the first row of the table refers to the first four bits of the output.

The selection of key bits in each round is determined by a simple key scheduling algorithm. The algorithm starts from a 64-bit secret key which includes 8 parity bits that are discarded after verification (the parity of each byte needs to be odd). The remaining 56 secret key bits are first permuted according to a permutation PC_1 (see Table 4). The result is split into two 28-bit words C_0 and D_0, which are cyclically rotated over 1 position to the left after rounds 1, 2, 9, 16, and over 2 positions after all other rounds (the rotated words are denoted by C_i and D_i). The round keys are constructed by repeatedly extracting 48 bits from C_i and D_i at 48 fixed positions determined by a table PC_2 (see Table 4). A convenient feature of this key scheduling algorithm is that the 28-bit words C_0 and D_0 are rotated over exactly 28 positions after 16 rounds. This allows hardware implementations to efficiently compute the round keys on-the-fly, both for the encryption and the decryption.

Table 1. Expansion E and permutation P

E						P			
32	1	2	3	4	5	16	7	20	21
4	5	6	7	8	9	29	12	28	17
8	9	10	11	12	13	1	15	23	26
12	13	14	15	16	17	5	18	31	10
16	17	18	19	20	21	2	8	24	14
20	21	22	23	24	25	32	27	3	9
24	25	26	27	28	29	19	13	30	6
28	29	30	31	32	1	22	11	4	25

CRYPTANALYSIS OF DES: DES has been subject to very intensive cryptanalysis. Initial attempts [16] did not identify any serious weaknesses except for the short key-size. It was noted that DES has a *complementation property*, i.e., given an encryption of the plaintext P into the ciphertext C under the secret key K: $E_K(P) = C$, one knows that the complement of the plaintext will be encrypted to the complement of the ciphertext under the complement of the key: $E_{\overline{K}}(\overline{P}) = \overline{C}$ (by complement we mean flipping of all the bits). Another feature was the existence of four weak keys, for which the cipher is an *involution*: $E_K(E_K(m)) = m$ (for these keys the contents of the key-schedule registers C and D is either all zeros or all ones), and six additional pairs of *semi-weak keys* for which $E_{K1}(E_{K2}(m)) = m$. The complementation and the weak-key properties are the result of interaction of the key-schedule, which splits the key-bits into two separate registers and the Feistel structure of the cipher. A careful study of the cycle structure of DES for weak and semi-weak keys has been given by Moore and Simmons [30]. See the book of Davies and Price [11] for a more detailed account on these and other features of DES identified prior to 1989. The properties of the group generated by DES permutations have also been studied intensively. Coppersmith and Grossman have shown [9] that in principle DES-like components can generate any permutation from the *alternating group* $A_{2^{64}}$ (all *even permutations*, i.e., those that can be represented with an even number of transpositions). However, DES implements only 2^{56} permutations, which is a tiny fraction of all the even permutations. If the set of 2^{56} DES permutations was closed under composition, then multiple encryption as used, for example in Triple-DES would be equivalent to single encryption and thus would not provide any additional strength. A similar weakness would be present if the size of the group generated by the DES permutations

Table 2. DES S-boxes

S_1 :	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0 :	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1 :	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2 :	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3 :	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S_2 :	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0 :	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
1 :	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
2 :	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
3 :	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S_3 :	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0 :	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
1 :	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
2 :	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
3 :	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S_4 :	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0 :	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
1 :	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
2 :	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3 :	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S_5 :	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0 :	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
1 :	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
2 :	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
3 :	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S_6 :	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0 :	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
1 :	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
2 :	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
3 :	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S_7 :	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0 :	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
1 :	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
2 :	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
3 :	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S_8 :	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0 :	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1 :	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
2 :	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
3 :	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Table 3. Initial and final permutations

IP								IP^{-1}							
58	50	42	34	26	18	10	2	40	8	48	16	56	24	64	32
60	52	44	36	28	20	12	4	39	7	47	15	55	23	63	31
62	54	46	38	30	22	14	6	38	6	46	14	54	22	62	30
64	56	48	40	32	24	16	8	37	5	45	13	53	21	61	29
57	49	41	33	25	17	9	1	36	4	44	12	52	20	60	28
59	51	43	35	27	19	11	3	35	3	43	11	51	19	59	27
61	53	45	37	29	21	13	5	34	2	42	10	50	18	58	26
63	55	47	39	31	23	15	7	33	1	41	9	49	17	57	25

Table 4. DES key schedule bit selections

PC_1							PC_2					
57	49	41	33	25	17	9	14	17	11	24	1	5
1	58	50	42	34	26	18	3	28	15	6	21	10
10	2	59	51	43	35	27	23	19	12	4	26	8
19	11	3	60	52	44	36	16	7	27	20	13	2
63	55	47	39	31	23	15	41	52	31	37	47	55
7	62	54	46	38	30	22	30	40	51	45	33	48
14	6	61	53	45	37	29	44	49	39	56	34	53
21	13	5	28	20	12	4	46	42	50	36	29	32

were small. Using the special properties of the weak keys it has been shown that DES generates a very large group, with a lower-bound of 2^{2499} permutations [7, 8], which is more than enough to make the *closure attacks* [18] impractical.

In the two decades since its design three important theoretical attacks capable of breaking the cipher faster than exhaustive search have been discovered: differential cryptanalysis (1990) [5], linear cryptanalysis (1993) [22], and the *improved Davies' attack* [3, 12]. An interesting twist is that differential cryptanalysis was known to the designers of DES and DES was constructed in particular to withstand[1] this powerful attack [8]. This explains why the cipher's design criteria were kept secret. Many of these secrets became public with the development of differential cryptanalysis and were later confirmed by the designers [33]. Both differential and linear attacks as well as Davies' attack are not much of a threat to real-life applications since they require more than 2^{40} texts for the analysis. For example: a linear attack requires 2^{43} known plaintexts to be encrypted under the same secret key. If the user changes the key every 2^{35} blocks the success probability of the attack would be negligible. Nevertheless, linear attacks were tested [23] in practice, run even slightly faster than theoretically predicted [17], and can potentially use twice less data in a chosen plaintext scenario [20]. In the case of the differential attack 2^{47} chosen plaintexts are required, though the attack would still work if the data is coming from up to 2^{33} different keys. However, the huge amount of *chosen* plaintext makes the attack impractical. In the case of Davies' attack the data requirement is 2^{50} *known plaintexts*, which is clearly impractical.

Although differential and linear attacks are hard to mount on DES, they proved to be very powerful tools for cryptanalysis; many ciphers which were not designed to withstand these attacks have been broken, some even with practical attacks. See for example the cipher FEAL [28, 29, 34]. In fact both attacks have been discovered while studying this cipher [4, 25], which was proposed as a more secure alternative to DES.

Exhaustive key *search* currently remains the biggest threat to the security of DES [31]. It was clear from the very beginning that a 56-bit key can be found in practical time by using a practical amount of resources. In 1977 a design for a key-search machine was proposed by Diffie and Hellman [13] with a cost of US$ 20 million and the ability to find a solution in a single day. Later Hellman proposed a chosen plaintext time-memory tradeoff approach, which would allow to build an even cheaper machine, assuming that

[1] Note that DES is strong but not optimal against linear cryptanalysis or improved Davies' attack, for example simple reordering of the S-boxes would make the cipher less vulnerable to these attacks without spoiling its strength against the differential attack [24]. This could indicate that the designers of DES did not know about such attacks.

a precomputation of 2^{56} encryption steps is done once for a single chosen plaintext. An effective and complete ASIC design for a key-search machine has been proposed by Wiener in 1993 [38]. It was shown that the US$ 1 million machine would run through the full key-space in 7 hours. It became clear in 1993 that DES had to be upgraded to triple-DES or be replaced; however NIST decided to reconfirm the FIPS standard a second time in 1993 for another five years (as FIPS 46-2). In 1998 the Electronic Frontier Foundation (EFF) built a working dedicated hardware machine which cost less than US$ 250,000 and could run through the full key-space in four days [14]. In a parallel development it was shown that a network of tens of thousands of PCs (a computational power easily available to a computer virus, for example) could do the same work in several weeks. At that time the *AES competition* had been started. As a result of this effort DES has been replaced by a successor, AES, which is based on a 128-bit block 128/192/256-bit key cipher Rijndael/AES.

EXTENSIONS OF DES: So where is DES today? DES is not obsolete. Due to substantial cryptanalytic effort and the absence of any practical cryptanalytic attack, the structure of DES has gained public trust. There have been several proposals to remedy the short key size problem plaguing the cipher:

- **Triple-DES (Diffie–Hellman [13]).** The idea is to multiple encrypt the block using DES three times with two or three different keys. This method gains strength both against cryptanalytic attacks as well as against exhaustive search. It is weak against related key attacks, however, and the speed is three times slower than single DES [31]. A two-key variant in the form of *Encrypt-Decrypt-Encrypt (E-D-E)*, i.e., $E_{K_1}(D_{K_2}(E_{K_1}(m)))$ has been proposed by IBM (Tuchman, 1978) and is still in wide use by the banking community. The convenience of this option is that it is backward compatible with a single DES encryption, if one sets $K_1 = K_2$.
- **Independent subkeys (Berson [1]).** The idea is to use independently generated 48-bit subkeys in each round. The total key-size is 768 bits, which stops the exhaustive search attack. However, the cryptanalytic attacks like differential or linear do work almost as good as for DES [31]. The speed of this proposal is as for single DES, but it has a slower key-schedule.
- **Slow key-schedule (Quisquater et al. [32] or Knudsen [10]).** Exhaustive search is stopped by loosing key-agility of a cipher.
- **DES-X (Rivest, 1984).** The idea is to XOR additional 64-bits of secret key material at the input and at the output of the cipher. See the article on DES-X for more details. This is very effective against exhaustive search, but does not stop old cryptanalytic attacks on DES, and allows new related key attacks. This approach allows the reuse of old hardware. The speed is almost the same as that of a single DES.
- **Key-dependent S-boxes (Biham-Biryukov [2]).** The idea is similar to DES-X, but the secret key material is XORed before and after the S-boxes. S-boxes are reordered to gain additional strength. The result is secure against exhaustive search and improves the strength against cryptanalytic attacks (with the exception of related key attacks). This approach applies to software or to hardware which permits the loading of new S-boxes. The speed is the same as that of a single DES.

As of today two-key and three-key triple DES is still in wide use and is included in NIST (FIPS 46-3, the 1999 edition [31]) and ISO standards. However, two-key triple DES variants are not recommended for use due to dedicated meet-in-the-middle attack by Oorschot and Wiener [37] with complexity $2^{120-\log n}$ steps given $O(n)$ *known plaintexts* and memory. For example, if $n = 2^{40}$, complexity of attack is 2^{80} steps. This attack is based on an earlier attack by Merkle and Hellman [27] which required 2^{56} *chosen plaintexts*, steps, and memory. These attacks are hard to mount in practice, but they are an important certificational weakness.

The recommended usage mode for triple-DES is *Encrypt-Encrypt-Encrypt (E-E-E)* (or *Encrypt-Decrypt-Encrypt (E-D-E)*) with three independently generated keys (i.e. 168 key bits in total), for which the best attacks are the classical meet-in-the-middle attack with only three known plaintexts, 2^{56} words of memory and 2^{111} analysis steps; and the attack by Lucks [21] which requires 2^{108} time steps and 2^{45} known plaintexts. These attacks are clearly impractical.

The DES-X alternative is also in popular use due to its simplicity and almost no speed loss. Thorough analysis of a generic construction is given in [19] and the best currently known attack is a slide attack [6] with complexity of n known plaintexts and $2^{121-\log n}$ analysis steps (for example: 2^{33} known plaintexts and memory and 2^{87} analysis steps).

Alex Biryukov
Christophe De Cannière

References

[1] Berson, T.A. (1983). "Long key variants of DES." *Advances in Cryptology—CRYPTO'82*, Lecture Notes in Computer Science, eds. D. Chaum, R.L. Rivest, and A.T. Sherman. Plenum Press, New York, 311–313.

[2] Biham, E. and A. Biryukov (1995). "How to strengthen DES using existing hardware." *Advances in Cryptography—ASIACRYPT'94*, Lecture Notes in Computer Science, vol. 917, eds. J. Pieprzyk and R. Safavi-Naini. Springer-Verlag, Berlin, 395–412.

[3] Biham, E. and A. Biryukov (1997). "An improvement of Davies' attack on DES." *Journal of Cryptology*, 10 (3), 195–206.

[4] Biham, E. and A. Shamir. "Differential cryptanalysis of DES-like cryptosystems." In Menezes and Vanstone [26], 2–21.

[5] Biham, E. and A. Shamir (1993). "Differential cryptanalysis of the data encryption standard." *Advances in Cryptology—CRYPTO'90*, eds. A.J. Menezes and S.A. Vanstone. Lecture Notes in Computer Science, vol. 537. Springer-Verlag, Berlin, 2–21.

[6] Biryukov, A. and D. Wagner (2000). "Advanced slide attacks." *Advances in Cryptology—EUROCRYPT 2000*, Lecture Notes in Computer Science, vol. 1807, ed. B. Preneel. Springer-Verlag, Berlin, 589–606.

[7] Campbell, K.W. and M.J. Wiener (1993). "DES is not a group." *Advances in Cryptology—CRYPTO'92*, Lecture Notes in Computer Science, vol. 740, ed. E.F. Brickell. Springer-Verlag, Berlin, 512–520.

[8] Coppersmith, Don (1994). "The data encryption standard (DES) and its strength against attacks." *IBM Journal of Research and Development*, 38 (3), 243–250.

[9] Coppersmith, D. and E. Grossman (1975). "Generators for certain alternating groups with applications to cryptography." *SIAM Journal Applied Math*, 29 (4), 624–627.

[10] Damgard, I. and L.R. Knudsen (1998). "Two-key triple encryption." *Journal of Cryptology*, 11 (3), 209–218.

[11] Davies, D.W. and W.L. Price (1989). *Security for Computer Networks* (2nd ed.). John Wiley & Sons, New York.

[12] Davies, D.W. and S. Murphy (1995). "Pairs and triplets of DES S-Boxes." *Journal of Cryptology*, 8 (1), 1–25.

[13] Diffie, W. and M. Hellman (1997). "Exhaustive cryptanalysis of the NBS data encryption standard." *Computer*, 10 (6), 74–84.

[14] Electronic Frontier Foundation (EFF) (1998). "DES cracker." http://www.eff.org/DEScracker/

[15] Feistel, H. (1973). "Cryptography and computer privacy." *Scientific American*, 228, 15–23.

[16] Hellman, M.E., R. Merkle, R. Schroppel, L. Washington, W. Diffe, S. Pohlig, and P. Schweitzer (1976). "Results of an initial attempt to cryptanalyze the NBS Data Encryption Standard." Technical report, Stanford University, USA.

[17] Junod, P. (2001). "On the complexity of Matsui's attack." *Selected Areas in Cryptography, SAC 2001*, Lecture Notes in Computer Science, vol. 2259, eds. S. Vaudenay and A.M. Youssef. Springer-Verlag, Berlin, 199–211.

[18] Kaliski, B.S., R.L. Rivest, and A.T. Sherman (1988). "Is the data encryption standard a group?" *Journal of Cryptology*, 1 (1), 3–36.

[19] Kilian, J. and P. Rogaway (1996). "How to protect DES against exhaustive key search." *Advances in Cryptology—CRYPTO'96*, Lecture Notes in Computer Science, vol. 1109, ed. N. Koblitz. Springer-Verlag, Berlin, 252–267.

[20] Knudsen, L.R. and J.E. Mathiassen (2001). "A chosen-plaintext linear attack on DES." *Fast Software Encryption, FSE 2000*, Lecture Notes in Computer Science, vol. 1978, ed. B. Schneier. Springer-Verlag, Berlin, 262–272.

[21] Lucks, S. (1998). "Attacking triple encryption." *Fast Software Encryption, FSE'98*, Lecture Notes in Computer Science, vol. 1372, ed. S. Vaudenay. Springer-Verlag, Berlin, 239–257.

[22] Matsui, M. (1993). "Linear cryptanalysis method for DES cipher." *Advances in Cryptology—EUROCRYPT'93*, Lecture Notes in Computer Science, vol. 765, ed. T. Helleseth. Springer-Verlag, Berlin, 386–397.

[23] Matsui, M. (1994). "The first experimental cryptanalysis of the data encryption standard." *Advances in Cryptology—CRYPTO'94*, Lecture Notes in Computer Science, vol. 839, ed. Y. Desmedt. Springer-Verlag, Berlin, 1–11.

[24] Matsui, M. (1995). "On correlation between the order of S-boxes and the strength of DES." *Advances in Cryptology—EUROCRYPT'94*, Lecture Notes in Computer Science, vol. 950, ed. A. De Santis. Springer-Verlag, Berlin, 366–375.

[25] Matsui, M. and A. Yamagishi (1992). "A new method for known plaintext attack of FEAL cipher." *Advances in Cryptology—EUROCRYPT'92*, Lecture Notes in Computer Science, vol. 658, ed. R.A. Rueppel. Springer-Verlag, Berlin, 81–91.

[26] Menezes, A. and S.A. Vanstone (eds.) (1991). *Advances in Cryptology—CRYPTO'90*, Lecture Notes in Computer Science, vol. 537, eds. A.J. Menezes and S.A. Vanstone. Springer-Verlag, Berlin.

[27] Merkle, R.C. and M.E. Hellman (1981). "On the security of multiple encryption." *Communications of the ACM*, 14 (7), 465–467.

[28] Miyaguchi, S. (1990). "The FEAL-8 cryptosystem and a call for attack." *Advances in Cryptology—CRYPTO'89*, Lecture Notes in Computer Science, vol. 435, ed. G. Brassard. Springer-Verlag, Berlin, 624–627.

[29] Miyaguchi, S. "The FEAL cipher family." In Menezes and Vanstone (26), 627–638.

[30] Moore, J.H. and G.J. Simmons (1987). "Cycle structures of the DES with weak and semi-weak keys."

Advances in Cryptology—CRYPTO'86, Lecture Notes in Computer Science, vol. 263, ed. A.M. Odlyzko. Springer-Verlag, Berlin, 9–32.

[31] National Institute of Standards and Technology (1979). "FIPS-46: Data Encryption Standard (DES)." Revised as FIPS 46-1:1988, FIPS 46-2:1993, FIPS 46-3:1999, available at http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf

[32] Quisquater, J.-J., Y. Desmedt, and M. Davio (1986). "The importance of "good" key scheduling schemes (how to make a secure DES scheme with $\leq$ 48 bit keys)." *Advances in Cryptology—CRYPTO'85*, Lecture Notes in Computer Science, vol. 218, ed. H.C. Williams. Springer-Verlag, Berlin, 537–542.

[33] sci.crypt (1992). "Subject: DES and differential cryptanalysis." Unpublished, http://www.esat.kuleuven.ac.be/~abiryuko/coppersmith_letter.txt

[34] Shimizu, A. and S. Miyaguchi (1998). "Fast data encipherment algorithm FEAL." *Advances in Cryptology—EUROCRYPT'87*, Lecture Notes in Computer Science, vol. 304, eds. D. Chaum and W.L. Price. Springer-Verlag, Berlin, 267–278.

[35] Smid, M. and D. Branstad (1998). "The data encryption standard: past and future." *Proceedings of the IEEE*, 76 (5), 550–559.

[36] Smith, J.L. (1971). "The design of Lucifer: A cryptographic device for data communications." Technical Report, IBM T.J. Watson Research Center, Yorktown Heights, NY, USA.

[37] van Oorschot, P.C. and M.J. Wiener (1990). "A known plaintext attack on two-key triple encryption." *Advances in Cryptology—EUROCRYPT'90*, Lecture Notes in Computer Science, vol. 473, ed. I. Damgård. Springer-Verlag, Berlin, 318–325.

[38] Wiener, M. (1996)."Efficient des key search." *Practical Cryptography for Data Internetworks*, presented at the rump session of CRYPTO'93, 31–79.

DATA REMANENCE

Data remanence is the ability of computer memory to retain previously stored information beyond its intended lifetime. With many data storage techniques, information can be recovered using specialized techniques and equipment even after it has been overwritten. Examples:

- Write heads used on exchangeable media (e.g., floppy disks, magstripe cards) differ slightly in position and width due to manufacturing tolerances. As a result, one writer might not overwrite the entire area on a medium that had previously been written to by a different device. Normal read heads will only give access to the most recently written data, but special high-resolution read techniques (e.g., magnetic-force microscopy) can give access to older data that remains visible near the track edges.
- Even with a perfectly positioned write head, the hysteresis properties of ferromagnetic media can result in a weak form of previous data to remain recognizable in overwritten areas. This allows the partial recovery of overwritten data, in particular with older low-density recording techniques. Protection measures that have been suggested in the literature against such data remanence include encryption, multiple overwriting of sensitive data with alternating or random bit patterns, the use of special demagnetization ("degaussing") equipment, or even the physical destruction (e.g., shredding, burning) of media at the end of its life time.
- The CMOS flip-flop circuits used in static RAM have been observed to retain data for minutes, at low temperatures in some cases even for hours, after the supply voltage has been removed [4]. The data remanence of RAM can potentially be increased where memory cells are exposed to constant data for extended periods of time or in the presence of ionizing radiation ("burn in"). Protection measures that are used in some commercial security modules include sensors for low temperature and ionizing radiation. These have to be connected to battery-powered alarm circuits that purge security RAM instantly in unusual environments. Another protection technique inverts or rotates bit patterns every few seconds, to avoid long-term exposure of memory cells to a constant value ("RAM saver").
- File and database systems do not physically overwrite ("purge") data when it is deleted by the user, unless special data purging functions designed for security applications are used. When objects are deleted, normally their storage area is only marked as available for reallocation. This leaves deleted data available for recovery with special undelete software tools, until the time when the respective memory location is needed to store new data.

Markus Kuhn

References

[1] A guide to understanding data remanence in automated information systems. National Computer Security Center, NCSC-TG-025, United States Department of Defense, September 1991.

[2] Gutmann, Peter (2001). "Data remanence in semiconductor devices." *Proceedings of the 10th USENIX Security Symposium, Washington, DC, USA*, 13–17.

[3] Gutmann, Peter (1996). "Secure deletion of data from magnetic and solid-state memory." *Sixth*

USENIX Security Symposium Proceedings, San Jose, CA, 77–89.

[4] Skorobogatov, Sergei (2002). “Low temperature data remanence in static RAM.” Technical Report UCAM-CL-TR-536, University of Cambridge, Computer Laboratory.

DAVIES–MEYER HASH FUNCTION

The Davies–Meyer hash function is a construction for a hash function based on a block cipher, where the length in bits of the hash result is equal to the block length of the block cipher. A hash function is a cryptographic algorithm that takes input strings of arbitrary (or very large) length and maps these to short fixed length output strings. The Davies–Meyer hash function is an unkeyed cryptographic hash function which may have the following properties: preimage resistance, second preimage resistance and collision resistance; these properties may or may not be achieved depending on the properties of the underlying block cipher.

In the following, the block length and key length of the block cipher will be denoted with n and k respectively. The encryption with the block cipher E using the key K will be denoted with $E_K(\cdot)$.

The Davies–Meyer scheme is an iterated hash function with a compression function that maps $k + n$ bits to n bits:

$$H_i = E_{X_i}(H_{i-1}) \oplus X_i. \qquad (1)$$

By iterating this function in combination with MD-strengthening (see hash functions) one can construct a hash function based on this compression function; this hash function is known as the Davies–Meyer hash function. It has been shown by Black and Rogaway [1] that in the black-box cipher model, if $k \geq n$ finding a (second) preimage requires approximately 2^n encryptions and finding a collision requires approximately $2^{n/2}$ encryptions.

In order to achieve an acceptable security level against (2nd) preimage attacks, the block length n needs to be at least 80 bits (in 2004); for collision resistance, the block length should be at least 160 bits (in 2004). This means that this scheme should not be used with 64-bit block ciphers (e.g., CAST-128, Data Encryption Standard (DES), FEAL, GOST, IDEA, KASUMI/MISTY1); it should only be used for (2nd) preimage resistance with 128-bit block ciphers (e.g., Rijndael/AES, Camellia, CAST-256, MARS, RC6, TWOFISH, and SERPENT). Very few 256-bit block ciphers exist; one exception is the 256-bit version of RC6.

It is also important to note that a block cipher may have properties which pose no problem at all when they are used only for encryption, but which may result in the Davies–Meyer construction of the block cipher to be insecure [3, 4]. A typical example are the complementation property and weak keys of *DES*; it is also clear that the Davies–Meyer construction based on DES-X is highly insecure. The fact that the key is known to an opponent may also result in security weaknesses (e.g., differential attacks of Rijmen and Preneel [5]). Hirose defines a block cipher secure against a known plaintext attack for which the Davies–Meyer hash function is not 2nd preimage resistant [2].

Since there are very few block ciphers with a 256-bit block length, the Davies–Meyer construction is rarely used to obtain collision resistant hash functions. However, this construction is very popular in custom designed hash functions such as MD4, MD5, and the SHA family. Indeed, the compression functions of these hash functions are designed using an internal block cipher structure; the compression functions are made non-invertible by applying the Davies–Meyer construction to these internal block ciphers.

Bart Preneel

References

[1] Black, J., P. Rogaway, and T. Shrimpton (2002). “Black-box analysis of the block-cipher-based hash-function constructions from PGV.” *Advances in Cryptology—CRYPTO 2002*, Lecture Notes in Computer Science, vol. 2442, ed. M. Yung. Springer-Verlag, Berlin, 320–355.

[2] Hirose, S. (2002). “Secure block ciphers are not sufficient for one-way hash functions in the Preneel-Govaerts-Vandewalle model.” *Selected Areas in Cryptography*, Lecture Notes in Computer Science, vol. 2595, eds. K. Nyberg and H.M. Heys. Springer-Verlag, Berlin, 339–352.

[3] Preneel, B. (1993). “Analysis and design of cryptographic hash functions.” *Doctoral Dissertation*, Katholieke Universiteit Leuven.

[4] Preneel, B., R. Govaerts, and J. Vandewalle (1994). “Hash functions based on block ciphers: A synthetic approach.” *Advances in Cryptology—CRYPTO’93*, Lecture Notes in Computer Science, vol. 773, ed. D. Stinson. Springer-Verlag, Berlin, 368–378.

[5] Rijmen V. and B. Preneel (1995). “Improved characteristics for differential cryptanalysis of hash functions based on block ciphers.” *Fast Software Encryption*, Lecture Notes in Computer Science, vol. 1008, ed. B. Preneel. Springer-Verlag, Berlin, 242–248.

DC NETWORK

The DC-Network is a synchronous network protocol by which the participants can broadcast messages anonymously and unobservably (see anonymity). A DC-Network can achieve sender and recipient anonymity even against computationally unrestricted attackers. The DC-Network protocol itself requires a network with a broadcast service. It was invented by David Chaum in 1984 [2–4] (hence the name DC-Network) and was somewhat re-discovered by Dolev and Ostrovsky in [5]. Messages can be addressed to one or more intended participants by encrypting them with their respective public encryption keys.

The basic DC-Network protocol allows one participant at a time to broadcast a message. In order to allow each participant to send at any time, the broadcast channel of the basic DC-Network needs to be allocated in a randomized fashion to all requesting senders. This can be achieved by well known contention protocols such as (slotted) ALOHA [8].

Consider the basic DC-Network protocol of n participants: $P_1, P_2, \dots, P_n$. Messages are strings of k bits. As a preparation, all participants agree on pairwise symmetric keys, i.e., randomly chosen bitstrings of k bit length. Let us denote the key between P_i and P_j as $k_{i,j}$. Assume participant P_1 wants to send a message m anonymously to all other participants (*anonymous broadcast*). This can be achieved by the basic DC-Network protocol, which works as follows:

Compute partial sums: Each participant P_i ($1 \leq i \leq n$) computes the XOR sum s_i of all the keys $k_{i,j}$ ($1 \leq j \leq n$) it has exchanged with each other participant P_j ($j \neq i$), such that $s_i = \sum_{j \neq i} k_{i,j}$. Participant P_1 also adds his message m into his partial sum such that $s_1 = m + \sum_{j \neq 1} k_{i,j}$.

Broadcast partial sums: Each participant P_i broadcasts its partial sum s_i.

Compute global sum: Each participant P_i computes the global sum $s = \sum_{i=1}^{n} s_i = m + \sum_{i=1}^{n} \sum_{j \neq i} k_{i,j} = m$ in order to recover the message m. Note that because $k_{i,j} = k_{j,i}$, all the keys cancel out leaving only m standing out of the global sum.

The basic DC-Network protocol is computationally efficient but requires n reliable broadcasts for each message, and even more in case of resolving message collisions where two or more participants are sending their messages in the same round.

The basic DC-Network protocol runs on any network architecture. If all participants are honest, everyone obtains the message m. Chaum [4] has proved that the basic DC-Network protocol achieves sender anonymity and recipient anonymity even against computationally unrestricted attackers. However, the proof for recipient anonymity implicitly assumes that the partial sums are broadcast reliably, i.e., each message of an honest participant is broadcast to all participants without being modified [9].

DC-Network is the continued execution of the basic DC-Network protocol. In this case, unconditional sender anonymity can be maintained only by using fresh pairwise keys in each round, which is a similar situation as for the *one-time pad* (see key). Waidner has proposed to choose the pairwise keys for each round of the basic DC-Network protocol based on a pseudo-random number generator seeded with a selection of messages exchanged in previous rounds of the basic DC-Network protocol. This is more practical, but results in sender anonymity that holds only against computationally restricted attackers [9].

The core idea behind the DC-Network is to substantially involve more participants in each communication than just the intended sender and recipient in order to conceal their sending and receiving within the set of participants. This approach introduces an inevitable vulnerability in case not all of the participants honestly follow the protocol. In fact, the service of a DC-Network can be easily disrupted by one or more cheating participants, who either stop sending their partial sums or sending wrong partial sums or sending too many messages (denial-of-service attack). Disruptions of the DC-Network have been considered by Chaum [1], Bos and den Boer [4] and Waidner [9].

The *key graph* of a DC-Network is the graph where each participant is represented by a vertex and each pairwise key $k_{i,j}$ is represented by an edge connecting the vertices representing P_i and P_j. If the key graph is complete as in the example above, no coalition of non-senders except all of them together gains any information about who sent m. Less complete key graphs can be used in order to reduce the amount of pairwise keys. On the other hand, the less complete the key graph is chosen, the more vulnerable the basic DC-Network protocol is against cheating participants who may collude and exchange their views in and after the basic DC-Network protocol in order to strip away the honest participants' anonymity. Collusions of cheating participants can be represented in the key graph by eliminating their mutual pairwise keys. That is if P_i, P_j are cheating, then we remove the key $k_{i,j}$ from the key graph, which may lead to an unconnected graph. Any participant represented by an unconnected vertex is entirely stripped of its anonymity. Such a participant is fully observable by the collusion of cheating participants. It is worth noting that the key graph can be chosen independently of the underlying network topology (ring, star, bus, etc.).

Waidner points out in [9] that reliable broadcast is probably an unrealistic assumption because it cannot be achieved by cryptographic means alone as there is no byzantine agreement against computationally unrestricted active attackers who may arbitrarily control many participants [6]. Furthermore, Waidner has shown how to achieve recipient anonymity against computationally unrestricted active attackers by replacing the reliable broadcast by a fail-stop broadcast, where honest participants stop as soon as they receive inconsistent inputs. Fail-stop broadcast can be realized by $O(n)$ messages, each signed by an unconditionally secure authentication *code*, or more efficiently by a fail-stop signature [10].

Interestingly, no widely accepted formal definitions of sender and recipient anonymity in a network, i.e., continued transmission service, has come up yet. Thus, a fully formal treatment of DC-Network protocols is not possible to date. A new approach in this direction was proposed by Schneider and Sidiropoulos [7] based on the CSP process algebra (Communicating Sequential Processes).

Compared to MIX-Networks, DC-Networks achieve sender anonymity even against computationally unrestricted active attackers, while MIX networks only achieve sender anonymity against computationally restricted attackers.

Gerrit Bleumer

References

[1] Bos, Jurjen and Bert den Boer (1990). "Detection of disrupters in the DC protocol." *Advances in Cryptology—EUROCRYPT'89*, Lecture Notes in Computer Science, vol. 434, eds. J.-J. Quisquater and J. Vandewalle. Springer-Verlag, Berlin, 320–327.

[2] Chaum, David (1981). "Untraceable electronic mail, return addresses, and digital pseudonyms." *Communications of the ACM*, 24 (2), 84–88.

[3] Chaum, David (1986). "Showing credentials without identification—signatures transferred between unconditionally unlinkable pseudonyms." *Advances in Cryptology—EUROCRYPT'85*, Lecture Notes in Computer Science, vol. 219, ed. F. Pichler. Springer-Verlag, Berlin, 241–244.

[4] Chaum, David (1988). "The dining cryptographers problem: Unconditional sender and recipient untraceability." *Journal of Cryptology*, 1 (1), 65–75.

[5] Dolev, Shlomi and Rafail Ostrovsky (1997). "Efficient anonymous multicast and reception." *Advances in Cryptology—CRYPTO'97*, Lecture Notes in Computer Science, vol. 1294, ed. B.S. Kaliski. Springer-Verlag, Berlin, 395–409.

[6] Lamport, Leslie, Robert Shostak, and Marshall Pease (1982). "The Byzantine Generals problem." *ACM Transactions on Programming Languages and Systems*, 4 (3), 382–401.

[7] Schneider, Steve and Abraham Sidiropoulos, (1996). "CSP and anonymity." *ESORICS'96 (4th European Symposium on Research in Computer Security), Rome*, Lecture Notes in Computer Science, vol. 1146, ed. V. Lotz. Springer-Verlag, Berlin, 198–218.

[8] Tanenbaum, Andrew S. (1988). *Computer networks* (2nd ed.). Prentice-Hall, Englewood Cliffs.

[9] Waidner, Michael (1990). "Unconditional sender and recipient untraceability in spite of active attacks." *Advances in Cryptology—EUROCRYPT'89*, Lecture Notes in Computer Science, vol. 434, eds. J.-J. Quisquater and J. Vandewalle. Springer-Verlag, Berlin, 302–319.

[10] Waidner, Michael and Birgit Pfitzmann (1990). "The dining cryptographers in the disco: Unconditional sender and recipient untraceability with computationally secure serviceability." *Advances in Cryptology—EUROCRYPT'89*, Lecture Notes in Computer Science, vol. 434, eds. J.-J. Quisquater and J. Vandewalle. Springer-Verlag, Berlin, 690.

DEBRUIJN SEQUENCE

A k-ary deBruijn sequence of order n is a sequence of period k^n which contains each k-ary n-tuple exactly once during each period. DeBruijn sequences are named after the Dutch mathematician Nicholas deBruijn. In 1946 he discovered a formula giving the number of k-ary deBruijn sequences of order n, and proved that it is given by $((k-1)!)^{k^{n-1}} \cdot k^{k^{n-1}-n}$. The result was, however, first obtained more than 50 years earlier, in 1894, by the French mathematician C. Flye-Sainte Marie.

For most applications binary deBruijn sequences are the most important. The number of binary deBruijn sequences of period 2^n is $2^{2^{n-1}-n}$. An example of a binary deBruijn sequence of period $2^4 = 16$ is $\{s_t\} = 0000111101100101$. All binary 4-tuples occur exactly once during a period of the sequence. In general, binary deBruijn sequences are balanced, containing the same number of 0's and 1's in a period, and they satisfy many randomness criteria, although they may be generated using deterministic methods. They have been used as a source of pseudo-random numbers and in key-sequence generators of stream ciphers.

A deBruijn sequence can be generated by a nonlinear feedback function in n-variables. From the initial state $(s_0, s_1, \ldots, s_{n-1})$ and a nonlinear Boolean function $f(z_0, z_1, \ldots, z_{n-1})$ one can generate the sequence

$$s_{t+n} = f(s_t, s_{t+1}, \ldots, s_{t+n-1}), \quad \text{for } t = 0, 1, 2, \ldots.$$

This can be implemented using an n-stage nonlinear shift register. For example the binary deBruijn sequence above of period $16 = 2^4$ can be generated by $s_{t+4} = f(s_t, s_{t+1}, s_{t+2}, s_{t+3})$, using the initial state (0000) and the Boolean function

$$f(z_0, z_1, z_2, z_3) = 1 + z_0 + z_1 + z_1 z_2 z_3.$$

The binary *deBruijn graph* B_n of order n is a directed graph with 2^n nodes, each labeled with a unique binary n-tuple and having an edge from node $S = (s_0, s_1, \ldots, s_{n-1})$ to $T = (t_0, t_1, \ldots, t_{n-1})$ if and only if $(s_1, s_2, \ldots, s_{n-1}) = (t_0, t_1, \ldots, t_{n-2})$. The successive n-tuples in a deBruijn sequence therefore form a *Hamiltonian cycle* in the deBruijn graph, meaning that a full cycle visits each node exactly once.

There are many algorithms for constructing deBruijn sequences. The following is perhaps one of the easiest to describe. Start with n zeros and append a one whenever the n-tuple thus formed has not appeared in the sequence so far, otherwise append a zero. The sequence of length $2^4 = 16$ above is an example of a deBruijn sequence constructed in this way. It is known that the decision of which bit to select next can be based on local considerations and storage requirements can be reduced to only $3n$ bits.

Any Boolean function f such that the mapping

$$(z_0, z_1, \ldots, z_{n-1}) \to (z_1, z_2, \ldots, z_{n-1}, f(z_0, z_1, \ldots, z_{n-1}))$$

is a permutation of the set of binary n-tuples is called a *nonsingular* Boolean function. It can be written in the form,

$$f(z_0, z_1, \ldots, z_{n-1}) = z_0 + g(z_1, z_2, \ldots, z_{n-1}) \pmod 2.$$

The truth table of a Boolean function $f(z_0, z_1, \ldots, z_{n-1})$ is a list of the values of $f(z_0, z_1, \ldots, z_{n-1})$ for all binary n-tuples. The weight of the truth table of f is the number of ones in this list.

Large classes of deBruijn sequences can be constructed by starting with a nonsingular Boolean function f that decomposes the deBruijn graph into several shorter disjoint cycles and then joining the cycles one by one until one arrives at a deBruijn sequence. To join two cycles one can find an n-tuple $(z_0, z_1, \ldots, z_{n-1})$ on a cycle (where we have $(z_1, z_2, \ldots, z_{n-1}, f(z_0, z_1, \ldots, z_{n-1}))$ on the same cycle) and $(z_1, z_2, \ldots, z_{n-1}, 1 + f(z_0, z_1, \ldots, z_{n-1}))$ on a different cycle. Then the two cycles will be joined after changing(complementing) $g(z_1, z_2, \ldots, z_{n-1})$ (leading to two changes of the truth table of f).

One common starting function is the nonsingular function corresponding to $g = 0$, i.e., $f(z_0, z_1, \ldots, z_{n-1}) = z_0$, that is known to decompose B_n into the Pure Circulating Register(PCR), consisting of all cycles of period dividing n. This is known to contain $Z(n) = \frac{1}{n} \sum_{d|n} \phi(d) 2^{n/d}$ cycles. For $n = 4$ the PCR consists of the cycles (0), (1), (01), (0001), (0011), and (0111). Another popular starting function is the Complementary Circulating Register(CCR) corresponding to $g = 1$, i.e., $f(z_0, z_1, \ldots, z_{n-1}) = z_0 + 1 \pmod 2$. This is known to contain $Z^*(n) = \frac{1}{2} Z(n) - \frac{1}{2n} \sum_{2d|n} \phi(2d) 2^{n/2d}$ cycles.

Another method to construct deBruijn sequences is to use recursive algorithms. There exist algorithms that take as input two deBruijn sequences of period 2^{n-1} and produce a deBruijn sequence of period 2^n.

The linear complexity of a deBruijn sequence is defined as the length of the shortest linear shift register that can be used to generate the sequence. The linear complexity L of a binary deBruijn sequence of period 2^n, $n \geq 3$, satisfies the double inequality,

$$2^{n-1} + n \leq L \leq 2^n - 1.$$

There exist deBruijn sequences that meet the upper and lower bounds with equality.

The *quadratic complexity* of a deBruijn sequence is the length of the shortest shift register that generates the sequence where the feedback function f is allowed to have quadratic terms. The quadratic complexity Q of a binary deBruijn sequence of period 2^n, $n \geq 3$, satisfies the double inequality

$$n + 2 \leq Q \leq 2^n - \binom{n}{2} - 1.$$

It is known that for any nonsingular Boolean function f, the number of cycles that it decomposes B_n into has the same parity as the weight of the truth table of g. Therefore for a deBruijn sequence the truth table of g has odd weight. It is further known that for a deBruijn sequence, the weight w of the truth table of g obeys,

$$Z(n) - 1 \leq w \leq 2^{n-1} - Z^*(n) + 1.$$

The lower bound can be achieved by starting with the PCR and joining cycles one at a time until we arrive at a deBruijn sequence. Each joining step will in this case increase the weight of the truth table of g by 1. Similarly we can construct deBruijn sequences of maximal weight by starting with the CCR and joining the cycles one by one, each joining step will in this case reduce the weight of the truth table of g by 1. For values $n < 7$ the number of deBruijn sequences of each possible weight of the truth table of g is known.

Tor Helleseth

References

[1] Fredricksen, H. (1982). "A survey of full length nonlinear shift register cycle algorithms." *SIAM Review*, 24 (2), 195–221.

[2] Golomb, S.W. (1982). *Shift Register Sequences*. Aegean Park Press, Laguna Hills, CA.

DECISIONAL DIFFIE–HELLMAN ASSUMPTION

The difficulty in computing discrete logarithms in some large finite groups has been the basis for many cryptographic schemes and protocols in the past decades, starting from the seminal Diffie–Hellman key agreement protocol [8], and continuing with encryption and digital signature schemes with a variety of security properties, as well as protocols for numerous other applications. Ideally, we would have liked to prove unconditional statements regarding the computational difficulty in computing discrete logarithms. However, since the current state of knowledge does not allow us to prove such claims, we formulate instead mathematical *assumptions* regarding the computational difficulty of this set of problems, and prove properties of the protocols we develop based on these assumptions.

A first assumption that is closely related to the Diffie–Hellman key exchange is the Computational Diffie–Hellman assumption (see Diffie–Hellman problem for more detail):

The Computational Diffie–Hellman (CDH) Problem: Given a group G, a generator g of G, and two elements $a = g^x, b = g^y \in G$, where x and y are unknown, compute the value $c = g^{xy} \in G$.

The Computational Diffie–Hellman (CDH) Assumption: Any probabilistic polynomial time algorithm solves the CDH problem only with negligible probability.

Notes:

(1) The probability is taken over the random choices of the algorithm. The probability is said to be negligible if it decreases faster than any inverse polynomial in the length of the input.

(2) As usual, the algorithm must run in time that is polynomial in the length of its input, namely in time that is polylogarithmic in the size of G. Also, a solution to the CDH problem is an algorithm that works for *all inputs.* Thus, the CDH assumption implies that there exists an infinite sequence of groups G for which no polytime algorithm can solve the CDH problem with probability that is not negligible. (Still, it is stressed that there exist infinite families of groups for which the CDH problem is in fact easy.)

(3) The assumption can be made with respect either to uniform-complexity or non-uniform complexity algorithms (i.e., circuit families.)

Indeed, the CDH assumption is very basic in cryptography. However, in many cases researchers were unable to prove the desired security properties of protocols based on the CDH assumption alone. (A quintessential example is the Diffie–Hellman key exchange protocol itself.) Furthermore, it appears that, at least in some groups, the CDH assumption captures only a mild flavor of the intractability of the Diffie–Hellman problem. Therefore the *Decisional* Diffie–Hellman assumption was formulated, as follows:

The Decisional Diffie–Hellman (DDH) Problem: Given a group G, a generator g of G, and three elements $a, b, c \in G$, decide whether there exist integers x, y such that $a = g^x$, $b = g^y$, and $c = g^{xy}$.

The Decisional Diffie–Hellman (DDH) Assumption (Version I): Any probabilistic polynomial time algorithm solves the DDH problem only with negligible probability.

The above formulation of the DDH assumption treats the problem as a worst-case computational problem (that is, an algorithm that solves the problem must work on *all* inputs. This formalization provides a useful comparison with the CDH problem. A much more useful alternative formulation of the DDH assumption only discusses the case where the inputs are taken from certain distributions. It is stated as follows:

The Decisional Diffie–Hellman (DDH) Assumption (Version II): The following two distributions are computationally indistinguishable:

- G, g, g^x, g^y, g^{xy}
- G, g, g^x, g^y, g^z

where g is a generator of group G and x, y, z are chosen at random from $\{1, \ldots, |G|\}$.

Note: More formally, the above two distributions are actually two distribution *ensembles,* namely two families of distributions where each distribution in a family is parameterized by the group G and the generator g. Recall that two distribution ensembles are computationally indistinguishable if, given a set of parameters (in our case, given G and g), no polytime algorithm can tell whether its input is drawn from the first ensemble or from the second. See more details in [10].

This version is useful since it asserts that, even when g^x and g^y are known, the value g^{xy} appears to be a "freshly chosen" random and independent number for any computationally bounded attacker. This holds in spite of the fact that the value g^{xy} is uniquely determined by g^x and g^y, thus its "entropy" (in the information-theoretic sense) is in fact zero. As shown in [12,14], the two versions of the DDH assumption are equivalent. (Essentially, equivalence holds due to the *random self reducibility* property of the discrete logarithm problem.)

Clearly, the DDH assumption implies the CDH assumption. Furthermore, it appears to be considerably stronger. In fact, there are groups where DDH is clearly false, but CDH may still hold. Still, there exist groups where DDH is believed to hold, for instance multiplicative groups of large prime order. A quintessential example is the subgroup of size q of $\mathbf{Z}_p^*$ (see modular arithmetic) where $p = 2q + 1$ and p, q are primes. (In this case the larger prime p is called a safe prime, and the smaller prime q is called a Sophie-Germain prime.)

Note: To see an example of a family of groups where DDH does not hold but CDH may still hold, consider a group G where it is easy to check whether an element is a quadratic residue (e.g., let $G = \mathbf{Z}_p^*$ where p is prime and $|\mathbf{Z}_p^*| = p - 1$ is even). Here, the CDH assumption may hold, yet DDH is false: If the input is drawn from G, g, g^x, g^y, g^{xy} then it is never the case that the last element is a quadratic non-residue but the preceding two elements are quadratic residues. In contrast, if the input is taken from G, g, g^x, g^y, g^z then the above event happens with significant probability. Other examples of such groups also exist. Here let us mention in particular the case of bilinear and multilinear pairings in Elliptic-Curve groups, which have been recently shown to be useful in cryptography. See identity based cryptosystem and for example [3].

SOME APPLICATIONS OF DDH: The DDH assumption proves to be very useful in cryptographic analysis of protocols. It is immediate to show based on DDH that the Diffie–Hellman key exchange results in a "semantically secure" key, i.e., a key that is indistinguishable from random. (It is not known how to prove this statement based on CDH alone.) Similarly, it implies the semantic security of ElGamal public key encryption. In addition, it is used in proving the security of efficient pseudorandom functions [12], chosen-ciphertext-secure encryption [6], commitment and zero-knowledge protocols [7,13], and many more.

VARIANTS OF DDH: The DDH assumption is only one of many assumptions that can be made on the intractability of the discrete logarithm problem. Several variants have been considered in the literature, some of which are *stronger* (allowing to prove stronger security properties of protocols), and some are *weaker* (and are believed to hold even in cases where DDH does not). Of the stronger ones, let us mention variants that allow the exponents x, y to be chosen from distributions other than uniform (or even in a semi-adversarial way) [5]. Other stronger variants are formalized in [3,9,11]. Of the weaker ones, we mention variants that give the distinguisher access only to a *hashed* version of the last element (either g^{xy} or g^z) e.g., [1].

BIBLIOGRAPHIC NOTE: The DDH assumption is implicit in many early works based on the Diffie–Hellman problem (starting with [8]). To the best of our knowledge, it was first formalized by Brands in [4] (in the context of undeniable signatures). It was further studied in [12,14] and is widely used since. For further reading, see Boneh's survey [2].

Ran Canetti

References

[1] Abdalla, M., M. Bellare, and P. Rogaway (2001). "DHIES: An encryption scheme based on the Diffie–Hellman problem." *Topics in Cryptology—CT-RSA 2001*, Lecture Notes in Computer Science, vol. 2020, ed. D. Naccache. Springer-Verlag, Berlin, 143–158.

[2] Boneh, Dan (1998). "The decision Diffie–Hellman problem." *Proceedings of the Third Algorithmic Number Theory Symposium*, Lecture Notes in Computer Science, vol. 1423, ed. J.P. Buhler. Springer-Verlag, Berlin, 48–63.

[3] Boneh, Dan and Alice Silverberg (2002). "Applications of multilinear forms to cryptography." *Proceedings of the Conferences in memory of Ruth Michler*, Contemporary Mathematics, American Mathematical Society. Cryptology ePrint Archive, Report 2002/080. Available on http://eprint.iacr.org/

[4] Brands, S. (1993). "An efficient off-line electronic cash system based on the representation problem." CWI TR CS-R9323.

[5] Canetti, R. (1997). "Toward realizing random oracles: Hash functions that hide all partial information." *Advances in Cryptology—CRYPTO'97,* Lecture Notes in Computer Science, vol. 1294, ed. B.S. Kaliski Jr. Springer-Verlag, Berlin, 455–469.

[6] Cramer, R. and V. Shoup (1998). "A practical public-key cryptosystem provably secure against adaptive chosen ciphertext attack." *Advances in Cryptology—CRYPTO'98*. Lecture Notes in Computer Science, vol. 1462, ed. H. Krawczyk. Springer-Verlag, Berlin, 13–25.

[7] Damgård, I. (2000). "Efficient concurrent zero-knowledge in the auxiliary string model." *Advances in Cryptography—EUROCRYPT 2000*, Lecture Notes in Computer Science, vol. 1807, ed. B. Preneel. Springer-Verlag, Berlin, 418–430.

[8] Diffie, W. and M. Hellman (1976). "New directions in cryptography." *IEEE Trans. Info. Theory*, IT-22, 644–654.

[9] Dodis, Yevgeniy (2002). "Efficient construction of (Distributed) verifiable random functions." Cryptology ePrint Archive, Report 2002/133. Available on http://eprint.iacr.org/

[10] Goldreich, O. (2001). *Foundations of Cryptography: Volume 1—Basic Tools*. Cambridge University Press, Cambridge.

[11] Lysyanskaya, Anna (2002). "Unique signatures and verifiable random functions from the DH-DDH separation." *Advances in Cryptology—CRYPTO 2002*, Lecture Notes in Computer Science, vol. 2442, ed. M. Yung. Springer-Verlag, Berlin, 597–612.

[12] Naor, Moni and Omer Reingold (1997). "Number-theoretic constructions of efficient pseudo-random functions." Extended abstract in *Proc. 38th IEEE Symp. on Foundations of Computer Science*, 458–467.

[13] Pedersen, T.P. (1991). "Distributed provers with applications to undeniable signatures." *Advances in Cryptography—EUROCRYPT'91*, Lecture Notes in Computer Science, vol. 547, ed. D.W. Davis. Springer-Verlag, Berlin, 221–242.

[14] Stadler, M. (1996). "Publicly verifiable secret sharing." *Advances in Cryptography—EUROCRYPT'96*, Lecture Notes in Computer Science, vol. 1070, ed. U. Maurer. Springer-Verlag, Berlin, 190–199.

DECRYPTION EXPONENT

The exponent d in the RSA private key (n, d). See RSA public key encryption.

Burt Kaliski

DENIABLE ENCRYPTION

Suppose Alice sends a message to Bob in an informal chat conversation. If a typical encryption scheme as the ElGamal public key encryption scheme or Rijndael/AES is used, an authority can ask Alice to reveal what she sent Bob. Indeed, in the case of ElGamal, when Alice sends $(C_1, C_2) = (g^r, my^r)$ and is forced to reveal her randomness r used, anybody can obtain m. So, one can view the ciphertext as some commitment to the message. In the case of AES, when Alice is forced to reveal the key she shares with Bob, the authority again can obtain the message. (Using zero-knowledge, Alice is not required to reveal the key.)

The goal of deniable encryption [1] is that Alice can send a private message to Bob, without having the ciphertext result in a commitment. This can be viewed as allowing her to deny having sent a particular message. A scheme satisfying this condition is called a *sender-deniable* encryption scheme.

There is a similar concern from Bob's viewpoint. Can Bob be forced to open the received ciphertext? Again if the ElGamal public key encryption scheme is used, then using his secret key, Bob can help the authority to decipher the message. So, Bob "cannot deny" having received the message. A scheme that solves this issue is called a *receiver-deniable* encryption scheme.

An example of a sender-deniable scheme explained informally, works as follows. Suppose the sender (Alice) and the receiver (Bob) have agreed on some pseudorandomness, such that both can distinguish it from true randomness. When Alice wants to send a message bit 1, she will send some pseudorandom string, otherwise she sends true randomness. Since the authority cannot distinguish the pseudorandom from the real random, Alice can pretend she sent the opposite bit of what she did. For further details, see [1].

Canetti–Dwork–Naor–Ostrovsky demonstrated that a sender-deniable encryption scheme can be transformed into a receiver-deniable one, as follows:

Step 1. The receiver (Bob) sends the sender (Alice) a random r using a sender-deniable encryption scheme.

Step 2. The sender Alice sends Bob the ciphertext $r \oplus m$, where $\oplus$ is the exor.

A receiver-deniable scheme can also be transformed into a sender deniable one, as explained in [1].

Yvo Desmedt

Reference

[1] Canetti, R., C. Dwork, M. Naor, and R. Ostrovsky (1997). "Deniable encryption." *Advances in Cryptology—CRYPTO'97, Proceedings* Santa Barbara, CA, USA, August 17–21 (Lecture Notes in Computer Science vol. 1294), ed. B.S. Kaliski, Springer-Verlag, Berlin, 90–104.

DENIAL OF SERVICE

In the most literal sense, whenever a legitimate principal is unable to access a resource for any reason, it can be said that a denial of service has occurred. In common practice, however, the term *Denial of Service* (DoS) is reserved only to refer to those times when an interruption in availability is the intended result of a deliberate attack [3]. Often, especially in the press, DoS is used in an even more narrow sense, referring specifically to remote flooding attacks (defined below) against network services such as web servers. When attempting to prevent access to a target service, the target itself can be attacked, or, equally effectively, another service upon which the target depends can be attacked. For example, to cause a DoS of a web server, the server program could be attacked, or the network connection to the server could be attacked instead.

DoS attacks can be categorized as either *local Denial of Service* attacks or *remote Denial of Service* attacks. Local DoS attacks are a type of privilege escalation, where a principal with legitimate access to a service is able to deny others access to it. In many older UNIX-like operating systems, for example, when a user goes to change their password, the system first locks the global password file before asking the user for their new password; until the user enters their new password, the file remains locked and no other users are able to change passwords. Remote DoS attacks, on the other hand, often require no special rights for the attacker, or are against services which do not require any authentication at all. Flooding a web server with millions of requests is an example of a common remote DoS attack.

Some DoS attacks, referred to as *logic* attacks in [7], work by exploiting programming bugs in the service being attacked, causing it to immediately exit or otherwise stop responding. Examples of these types of attacks include the Windows 95 Ping-of-Death, BIND nameserver exit-on-error attacks, and countless buffer overflow attacks which crash, but do not compromise,[1] services. These kinds of DoS attacks are the easiest to prevent, since the attack is caused by invalid behavior that would not be expected from legitimate principals. By fixing bugs and more carefully filtering out bad input, these types of DoS attacks can be prevented. The attacks are also very *asymmetric* however, making them very dangerous until all vulnerable services have been upgraded. With these attacks, very little effort on the part of the attacker (a single malformed message typically) leads to a complete Denial of Service. An attacker with limited resources is able to quickly cause a great deal of damage with these attacks.

In contrast, *flooding* DoS attacks work by consuming limited resources on the server. Resources commonly targeted by these attacks include memory, disk space, CPU, and network bandwidth. Simple local DoS attacks such as acquiring and never releasing a shared lock also fall into this group. With these attacks, the problem lies in the rate the attacker does something, not in what they do. These attacks take a normal, acceptable activity such as forking a new process or requesting a web page, and raise it to an attack by performing the activity to excess.

Because these attacks involve behavior that would normally be perfectly acceptable, there is typically no way to tell with certainty that a request to a service is part of an attack. In some cases, particularly local DoS attacks, the consumption of resources can be limited by placing caps on how much of the resource any single user can consume. Limits can be placed on how much memory, disk space, or CPU a single user can use, and timeouts can be set whenever an exclusive lock is given out. The difficulty with using limits to prevent these attacks is that if a user needs to exceed one of these caps for legitimate purposes, they are unable to; the solution to the first DoS attack causes a Denial of Service of a different kind. Because most solutions to flooding attacks rely on some heuristic to determine when behavior is malicious, there are always some false positives which cause the prevention to be a DoS itself.

As with logic attacks, some flooding attacks are also highly asymmetric. In particular, many standard protocols (such as IP, TCP (see firewall) and SSL/TLS (see Secure Socket Layer and Transport Layer Security)) allow for asymmetric attacks because they require the service to keep state or perform expensive computations for the attacker.

[1] Technically, if an attack's primary purpose is to compromise a service, and, as a side effect, it crashes the service, this is not considered a DoS attack [3].

If an attacker begins many protocol sessions but never completes them, resources can quickly be exhausted. TCP SYN flood and incomplete IP fragment attacks both work by exhausting available buffers within the server's networking stack. When beginning SSL/TLS sessions the server must perform CPU-intensive public key cryptography operations which take orders of magnitude longer than it takes an attacker to send a request. To remove the asymmetry of these attacks, techniques that reduce or remove the state the server must keep [6] or force the client to perform a comparable amount of computation [1] have been proposed.

Flooding attacks which have similar resource requirements for the attacker and victim comprise the final group of common DoS attacks. While the previous attacks described have had an element of skill to them, finding and exploiting some programming error or imbalance in protocol design, these attacks are nothing more than a shoving match, with the participant with the most of the resource in question winning. Smurf attacks and DNS flooding are well known examples of these brute-force DoS attacks.

Often, the attacker does not have an excess of the resource (usually network bandwidth) themselves. Instead, to carry out their attack they first compromise a number of other hosts, turning them into *zombies*, and then have their zombies attack the service simultaneously. This type of attack is known as a *Distributed Denial of Service* (DDoS) attack, and has proven very effective in the past against a number of popular and very well connected Internet servers such as Yahoo! and eBay.

With all types of remote flooding attacks, if the source of the flood can be identified, it can be blocked with minimal disruption to non-attack traffic. With DDoS attacks, this identification is the main difficulty, since no single zombie produces an exceptionally large number of requests. Blocking the wrong source results in a DoS itself. Further complicating identification, many attackers mask themselves by forging, or *spoofing*, the source of requests. Traceback techniques [2,8] can be used to identify the true source, but their accuracy degrades as more sources are present. Egress filtering, which blocks packets from leaving edge networks if they claim to have not originated from that network, can prevent spoofing. Unfortunately, all networks must employ egress filtering before it is an adequate solution. Since most DoS attacks employ spoofing, Backscatter analysis [7] actually takes advantage of it, looking at replies from victims to the spoofed sources to determine world-wide DoS activity.

Once the true source of a flood has been identified, filters can be installed to block the attack. With bandwidth floods in particular, this blocking may need to occur close to the attacker in the network in order to fully block the DoS. This can either be arranged manually, through cooperation between network administrators, or automatically through systems like Pushback [5].

As seen above, many Denial of Service attacks have no simple solutions. The very nature of openly accessible services on the Internet leaves them vulnerable from these attacks. It is an interesting and rapidly evolving type of security attack. The list of resources at [4] is updated periodically with pointers to new attacks and tools for protecting services, and makes a good starting point for further exploring the causes and effects of DoS attacks, and the state of the art techniques in dealing with them.

Eric Cronin

References

[1] Aura, T., P. Nikander, and J. Leiwo (2000). "DoS resistant authentication with client puzzles." *Proc. of the Cambridge Security Protocols Workshop 2000*.

[2] Bellovin, S. (2002). "ICMP traceback messages." *Internet Draft.* draft-ietf-itrace-02.txt

[3] CERT Coordination Center (2001). "Denial of service attacks." http://www.cert.org/tech_tips/denial_of_service.html

[4] Dittrich, D. "Distributed denial of service (DDoS) attacks/tools." http://staff.washington.edu/dittrich/misc/ddos/

[5] Ioannidis, J. and S.M. Bellovin (2000). "Implementing pushback: Router-based defense against DDoS attacks." *Proc. of NDSS 2002*.

[6] Lemon, J. (2001). "Resisting SYN flood DoS attacks with a SYN cache." http://people.freebsd.org/~jlemon/papers/syncache.pdf

[7] Moore, D., G.M. Voelker, and S. Savage (2001). "Inferring internet denial-of-service activity." *Proc. of the 10th USENIX Security Symposium*.

[8] Savage, S., D. Wetherall, A. Karlin, and T. Anderson (2000). "Practical network support for IP traceback." *Proc. of ACM SIGCOMM 2000*.

DERIVED KEY

A derived key is a key, which may be calculated (derived) by a well-defined algorithm from a input consisting of public as well as secret data. As an example, the initial secret data might be a random seed, i.e., a string of random bits (see modular arithmetic), which is then exponentiated modulo, e.g., an *RSA*-modulus (say both of length 1024; see RSA public key encryption), after which the derived key may be the lower 128 bits of the result R (current seed), which is kept and exponentiated again for the derivation of the next key. The advantage is that if two parties share the same initial seed, they may independently of each other calculate identical derived keys by keeping track of the number of iterations.

Peter Landrock

DESIGNATED CONFIRMER SIGNATURE

Designated confirmer signatures (or sometimes simply 'confirmer signatures') are digital signatures that can be verified only by some help of a semi-trusted designated confirmer. They were introduced by Chaum in [3] as an improvement of convertible undeniable signatures. Unlike an ordinary digital signature that can be verified by anyone who has access to the public verifying key of the signer (*universal verifiability*), a designated confirmer signature can only be verified by engaging in a—usually interactive—protocol with the designated confirmer. The outcome of the protocol is an affirming or rejecting assertion telling the verifier whether the signature has originated from the alleged signer or not.

The main difference to (convertible) undeniable signatures is that the capabilities to produce signatures and to confirm signatures are laid into different hands, which has several advantages. Designated confirmer signatures improve the availability and reliability of the confirmation services for verifiers. Verifiers can rely on a designated confirmer instead of having to rely on the signers themselves. The designated confirmer can be organized as one or more authorities with a higher availability than each signer can afford to provide, and the designated confirmer can provide confirmation services according to a clearly stated confirmation policy, which can also be subject to independent audit on a regular basis. In practice, a designated confirmer would conceivably contract multiple signers and provide confirmation services to all their respective verifiers. Another way of increasing the availability of the confirmation services is by using an undeniable signature scheme with distributed provers as proposed by Pedersen [7]. Another advantage of designated confirmer signatures is that they alleviate the problem of coercable signers. In undeniable signature schemes, the signer may be blackmailed or bribed to confirm or disavow an alleged signature. This may be harder to accomplish with a designated confirmer organized as an authority with proper checks and balances.

Designated confirmer signatures are a useful tool to construct protocols for contract signing [1]. The trusted third party in contract signing takes the role of a designated confirmer. Each participant produces a designated confirmer signature of his statement and distributes it to all other participants and to the trusted third party. After the trusted third party has collected the statements and corresponding designated confirmer signatures from all participants, it converts them into ordinary digital signatures and circulates them to all participants according to a predefined policy. Designated confirmer signatures are also useful to construct *verifiable signature sharing* schemes [4].

A designated confirmer signature scheme has three operations: (i) An operation for generating double key pairs, one key pair of a private signing key with a public verifying key and another key pair of a private confirmer key with a public confirmer key, (ii) an operation for signing messages, and (iii) a *confirming operation* for proving signatures valid (confirmation) or invalid (disavowal). The private signing key is known only to the signer, the private confirmer key is known only to the confirmer, and the public verifying key as well as the public confirmer key are publicly accessible through authenticated channels, e.g., through a public key infrastructure (PKI). The signing operation is between a signer using the private signing key and a verifier using the public verifying key. The verifying operation is between the designated confirmer using its private confirmer key and a verifier using the public confirmer key. Furthermore, there is (iv) an *individual conversion operation* for converting individual designated confirmer signatures into ordinary digital signatures, and (v) a *universal verifying operation* to verify such converted signatures.

The characteristic security requirements of a designated confirmer signature scheme are similar to those of a convertible undeniable signature scheme [2]:

Unforgeability: Resistance against existential forgery under adaptive chosen message attacks by computationally restricted attackers.

Invisibility: A cheating verifier, given a signer's public verifying key, public confirmer key, a message, a designated confirmer signature and oracle access to the signer, cannot decide with probability better than pure guessing whether the signature is valid for the message with respect to the signer's verifying key or not. (This implies non-coercibility as described above.)

Soundness: A cheating designated confirmer cannot misuse the verifying operation in order to prove a valid signature to be invalid (non-repudiation), or an invalid signature to be valid (false claim of origin).

Non-transferability: A cheating verifier obtains no information from the confirming operation that allows him to convince a third party that the alleged signature is valid or invalid, regardless if the signature is valid or not.

Validity of Conversion: A cheating designated confirmer with oracle access to a signer cannot fabricate a converted signature valid for a message m with respect to the signer's public verifying key unless that signer has produced a designated confirmer signature for m before.

Practical constructions have been proposed by Chaum [3], Okamoto [6], Michels and Stadler [5], and by Camenisch and Michels [2]. All of them propose an *individual conversion operation*, but none of them discusses a *universal conversion operation* analogous to that of convertible undeniable signatures. Michels and Stadler [5] have discussed designated confirmer signatures that can be converted into well known ordinary signatures such as RSA digital signatures, Schnorr digital signatures, Fiat and Shamir signatures, or ElGamal digital signatures.

Designated confirmer signatures are a relatively young concept, which have not yet been blended with other interesting types of signature schemes such as threshold signatures, group signatures, or fail-stop signatures.

Gerrit Bleumer

References

[1] Asokan, N., Victor Shoup, and Michael Waidner (1998). "Optimistic fair exchange of digital signatures." *Advances in Cryptology—EUROCRYPT'98*, Lecture Notes in Computer Science, vol. 1403, ed. K. Nyberg. Springer-Verlag, Berlin, 591–606.

[2] Franklin, Matthew K. and Michael K. Reiter (1995). "Verifiable signature sharing." *Advances in Cryptology—EUROCRYPT'95*, Lecture Notes in Computer Science, vol. 921, eds. L.C. Guillou and J.-J. Quisquater. Springer-Verlag, Berlin, 50–63.

[3] Chaum, David (1995). "Designated confirmer signatures." *Advances in Cryptology—EUROCRYPT'94*, Lecture Notes in Computer Science, vol. 950, ed. A. De Santis. Springer-Verlag, Berlin, 86–91.

[4] Camenisch, Jan and Markus Michels (2000). "Confirmer signature schemes secure against adaptive adversaries." *Advances in Cryptography—EUROCRYPT 2000*, Lecture Notes in Computer Science, vol. 1807, ed. B. Preneel. Springer-Verlag, Berlin, 243–258.

[5] Michels, Markus and Markus Stadler (1998). "Generic constructions for secure and efficient confirmer signature schemes." *Advances in Cryptology—EUROCRYPT'98*, Lecture Notes in Computer Science, vol. 1403, ed. K. Nyberg. Springer-Verlag, Berlin, 406–421.

[6] Okamoto, Tatsuaki (1994). "Designated confirmer signatures and public-key encryption are equivalent." *Advances in Cryptology—CRYPTO'94*, Lecture Notes in Computer Science, vol. 839, ed. Y.G. Desmedt. Springer-Verlag, Berlin, 61–74.

[7] Pedersen, Torben Pryds (1991). "Distributed provers with applications to undeniable signatures (Extended abstract)." *Advances in Cryptology—EUROCRYPT'91*, Lecture Notes in Computer Science, vol. 547, ed. D.W. Davies. Springer-Verlag, Berlin, 221–242.

DES-X (OR DESX)

DES-X is a 64-bit block cipher with a $2 \times 64 + 56 = 184$-bit key, which is a simple extension of DES (see Data Encryption Standard). The construction was suggested by Rivest in 1984 in order to overcome the problem of the short 56-bit key-size which made the cipher vulnerable to exhaustive key search attack. The idea is just to XOR a secret 64-bit key $K1$ to the input of DES and to XOR another 64-bit secret key $K2$ to the output of DES: $C = K2 \oplus DES_K(P \oplus K1)$. The keys $K1, K2$ are called *whitening keys* and are a popular element of modern cipher design. The construction itself goes back to the work of Shannon [6, p. 713], who suggested the use of a fixed mixing permutation whose input and output are masked by the secret keys. This construction has been shown to have provable security by Even–Mansour [2] if the underlying permutation is *pseudorandom* (i.e., computationally indistinguishable from a random permutation). A thorough study of DES-X was given in the work of Kilian–Rogaway [3], which builds on [2] and uses a blackbox model of security. Currently, the best attack on DES-X is a known-plaintext slide attack discovered by Biryukov–Wagner [1] which has complexity of $2^{32.5}$ known plaintexts and $2^{87.5}$ time of analysis. Moreover the attack is easily converted into a ciphertext-only attack with the same data complexity and 2^{95} offline time complexity. These attacks are mainly of theoretical interest due to their high time complexities. However, the attack is generic and would work for any cipher F used together with post- and pre-whitening with complexity $2^{(n+1)/2}$ known plaintexts and $2^{k+(n+1)/2}$ time steps (here n is the block size, and k is the keysize of the internal cipher F). A related key-attack on DES-X is given in [4]. Best conventional attack, which exploits the internal structure of DES, would be a linear cryptanalysis attack, using 2^{61} known plaintexts [3].

Alex Biryukov

References

[1] Biryukov, A. and D. Wagner (2000). "Advanced slide attacks." *Advances in Cryptology—EUROCRYPT 2000*, Lecture Notes in Computer Science, vol. 1807, ed. B. Preneel. Springer-Verlag, Berlin, 589–606.

[2] Even, S. and Y. Mansour (1997). "A construction of a cipher from a single pseudorandom permutation." *Journal of Cryptology*, 10 (3), 151–161. Springer-Verlag.

[3] Kaliski, B. and M. Robshaw (1996). "Multiple encryption: Weighing security and performance." *Dr. Dobb's Journal*, 243 (1), 123–127.

[4] Kelsey, J., B. Schneier, and D. Wagner (1997). "Related-key cryptanalysis of 3-WAY, Biham-DES, CAST, DES-X, NewDES, RC2, and TEA." *Proceedings of ICICS*, Lecture Notes in Computer Science, 1334, eds. Y. Han, T. Okamoto and S. Qing. Springer, Berlin, 233–246.

[5] Kilian, J. and P. Rogaway (1996). "How to protect against exhaustive key search." *Advances in Cryptology—CRYPTO'96*, Lecture Notes in Computer Science, vol. 1109, ed. N. Koblitz. Springer-Verlag, Berlin, 252–267.

[6] Shannon, C. (1949). "Communication theory of secrecy systems. A declassified report from 1945." *Bell Syst. Tech. J.* (28), 656–715.

DICTIONARY ATTACK (I)

Dictionary attack is an exhaustive cryptanalysis approach in which the attacker computes and stores a table of plaintext–ciphertext pairs $(P, C_i = E_{K_i}(P), K_i)$ sorted by the ciphertexts C_i. Here the plaintext P is chosen in advance among the most often encrypted texts like "login:", "Hello John", etc. and the key runs through all the possible keys K_i. If P is encrypted later by the user and the attacker observes its resulting ciphertext C_j, the attacker may search his table for the corresponding ciphertext and retrieve the secret key K_j.

The term dictionary attack is also used in the area of password guessing, but with a different meaning.

Alex Biryukov

DICTIONARY ATTACK (II)

A *dictionary attack* is a password [1] guessing technique in which the attacker attempts to determine a user's password by successively trying words from a *dictionary* (a compiled list of likely passwords) in the hope that one of these password guesses will be the user's actual password. In practice, the attacker's *dictionary* typically is not restricted to words from a traditional natural-language dictionary, but may include one or more of the following:

- variations on the user's first or last name, initials, account name, and other relevant personal information (such as address and telephone number, pet's name, and so on);
- words from various databases such as male and female names, places, cartoon characters, films, myths, and books;
- spelling variations and permutations of the above words, such as replacing the letter "o" with the number "0", using random capitalization, and so on;
- common word pairs.

Dictionary attacks can be quite successful in many environments because of the tendency of users to make poor password choices (unfortunately, passwords that are easily memorized by a legitimate user are also easily guessed by an attacker). These attacks can be performed in online mode (trying successive passwords until a login is successful) or offline mode (hashing or encrypting a dictionary of words and looking for any matches in a copied system file of hashed or encrypted user passwords). Server limits on the number of unsuccessful login attempts can help to thwart online attacks and the use of "salt" [see salt] can help to thwart offline attacks.

Carlisle Adams

References

[1] Schneier, B. (1996). *Applied Cryptography: Protocols, Algorithms, and Source Code in C* (2nd ed.). John Wiley & Sons, New York.

[2] Stallings, W. (1999). *Cryptography and Network Security: Principles and Practice* (2nd ed.). Prentice Hall.

DIFFERENTIAL CRYPTANALYSIS

Differential cryptanalysis is a general technique for the analysis of symmetric cryptographic primitives, in particular of block ciphers and hash functions. It was first publicized in 1990 by Biham and Shamir [3, 4] with attacks against reduced-round variants of the Data Encryption Standard (DES) [14], and followed in 1991 by the first attack against DES which was faster than exhaustive key search [6].

Let P be a plaintext, and let C be the corresponding ciphertext encrypted under the (unknown) key K, such that $C = E_K(P)$. Let P^* be a second plaintext, and let C^* be the corresponding ciphertext under the same (unknown) key K, $C^* = E_K(P^*)$. We define the difference of the plaintexts as $P' = P \oplus P^*$, and the difference of the ciphertexts as $C' = C \oplus C^*$. Also for any intermediate data X during encryption (for example, the data after the third round, or the input to some operation in the fifth round), let the corresponding data during the encryption of P^* be denoted by X^*, and let the difference be $X' = X \oplus X^*$.

Differential cryptanalysis studies the differences, usually by means of exclusive-or (XOR), as they evolve in the various rounds and various operations of the cipher. Linear and affine operations do not affect the differences, or affect the differences in a predictable way: bit-permutation operations (that reorder the bits of the data X to $P(X)$) reorder the differences in the same way (i.e., to $P(X') = P(X) \oplus P(X^*)$); selections (that select some of the bits of the data) also select the bits of the differences; and XOR operations of two values $X \oplus Y$, also XOR the differences of the values to $X' \oplus Y' = (X \oplus Y) \oplus (X^* \oplus Y^*)$. An important observation is that mixing subkeys into the data may be discarded by means of differences: if the mixing of subkeys to the data is performed using an XOR operation by $Y = X \oplus K$, then in the second encryption it is $Y^* = X^* \oplus K$, and the output difference of the key mixing is $Y' = Y \oplus Y^* = (X \oplus K) \oplus (X^* \oplus K) = X'$, which is independent of the subkey. Key mixings may thus be ignored in the predictions of the differences.

For non-linear operations (such as S boxes) we can also study the evolvement of the differences. Certainly, when the difference of the input is 0, the two inputs are equal, and thus also the two outputs are equal, having a difference 0 as well. When the input difference is nonzero, we cannot predict the output difference, as it may have many different output differences for any input difference. However, it is possible to predict statistical information on the output difference given the input difference. Take for example S box S1 of DES. This S box has 6 input bits and 4 output bits. For each input difference X' there are 64 possible pairs of inputs with this difference (for any possible input X, the second input is computed by $X^* = X \oplus X'$). These 64 pairs may have various output differences. The main observation is that the output differences are not distributed uniformly. For example, for the input difference 34_x (the subscript x denotes that the number is in hexadecimal notation), no pair has output difference 0, nor 5 nor 6, and several other output differences; two pairs have output difference 4, eight pairs have output difference 1, and 16 of the 64 pairs with this input difference have output difference 2. For this input difference, a cryptanalyst can thus predict with probability 1/4 that the output difference is 2. A *difference distribution table* of an S box (or operation) is a table that lists the number of pairs which fulfill the input and output differences for each possible input and output differences, where the rows denote all the possible input differences, the columns all the output differences, and each entry contains the number of pairs with the corresponding differences. In the example above, the difference distribution table of S1 of DES has value 16 in row 34_x column 2.

Differential cryptanalysis defines characteristics that describe possible evolvements of the differences through the cipher. Each characteristic has a plaintext difference for which it predicts the differences in the following rounds. A pair of plaintexts for which the differences of the plaintexts and the intermediate data (when encrypted under the used key) are exactly as predicted by the characteristic are called *right pairs* (all other pairs are called *wrong pairs*). The probability that a characteristic succeeds to predict the differences (i.e., that a random pair is a right pair, given that the plaintext difference is as required by the characteristic) depends on the probabilities induced by the input and output differences for each S box (or each operation), where the total probability is the product of the probabilities of the various operations (assuming that the probabilities are independent, which is usually the case; otherwise the product is usually a good approximation for the probability).

Given the expected difference for the intermediate data before the last round (or more generally in some round near the end of the cipher), it may be possible to deduce the unknown key by a statistical analysis. The attack is a chosen plaintext attack that is performed in two phases: In the *data collection phase* the attacker requests encryption of a large number of pairs of plaintexts, where the differences of all the plaintext pairs are selected to have the plaintext difference of the characteristic. In the data analysis phase the attacker then recovers the key from the collected ciphertexts.

Assume that the probability of the characteristic is p (i.e., a fraction p of the pairs are expected to be right pairs). It is then expected that for a fraction p of the pairs, the difference of the data before the last round is as predicted by the characteristic. An (inefficient) method for deriving the subkey of the last round is then to try all the possibilities of the subkey of the last round. For each possible subkey partially decrypt all the ciphertexts by one round, and for each pair compute the differences of the data before the last round, by XORing the data resulting from the partial decryptions. For wrong guesses of the subkey it is expected that the difference predicted by the characteristic appears rarely, and for the correct value of the subkey it is expected that this difference appears for a fraction p or more of the pairs (as there is a fraction of about p of right pairs that are assured to suggest this difference, and as wrong pairs may also suggest this difference). In particular, if the probability p is not too low, it is expected that the correct subkey is the one which gives the expected difference most frequently. It should be noted that the derivation of the last subkey is usually much more efficient than (but equivalent in results to) this described algorithm, using the information of the input and output differences for each S box (or operation) in the last round. It should also be noted that in many cases characteristics shorter by more than one round than the cipher (usually up to three rounds shorter) can also be used for differential attacks.

Differential cryptanalysis usually requires a small multiple of $1/p$ pairs of chosen plaintexts, when using a characteristic with probability p, in order to ensure that sufficiently many right pairs appear in the data. This amount of encrypted data may be very large (about 2^{47} chosen plaintext blocks in the case of DES), making the complexity of the data collection phase larger than the complexity of the data analysis phase in most cases. The large number of chosen plaintexts may by itself make the attack impractical, as it transfers the responsibility of computing the major part of the attack from the attacker to the attacked party, who is required to encrypt a large number of chosen plaintexts for the attacker to be able to mount his attack. It is therefore common in such cases to quote the complexity of a differential attack to be the number of required chosen plaintexts.

After the publication of the differential cryptanalysis attack on DES, whose complexity is 2^{47} (it requires 2^{47} chosen plaintexts and the time of analysis is less than 2^{40}), IBM announced that they were aware of differential cryptanalysis when they designed DES, and actually designed it to withstand differential attacks. Moreover, differential attacks (to which they called the *T method*) were classified as top secret for purposes of US national security, and IBM were requested by the NSA not to publish any information on them.

There are various improvements of differential cryptanalysis aimed to reduce the complexity of differential attacks. One simple method is a combination of several characteristics in a single larger structure. In case two characteristics are used, such a structure is called a *quartet*. It contains four plaintexts of the form P, $P \oplus \Omega_p^1$, $P \oplus \Omega_p^2$, $P \oplus \Omega_p^1 \oplus \Omega_p^2$, for the plaintexts differences Ω_p^1 and Ω_p^2. It can easily be seen that in such a quartet each difference appears twice: the first difference appears as the difference of the first two plaintexts, and also as the difference of the other two plaintexts; the second difference appears as the difference of the first and third plaintexts, and also as the difference of the second and fourth plaintexts. Thus, a total of four pairs are contained in a quartet; without using quartets only two pairs are contained in the same number of plaintexts. Larger structures of eight plaintexts using three different characteristics contain 12 pairs. Such structures are useful when there are several high-probability characteristics that can be used for an attack, as if the second best characteristic has a relatively low probability, the benefit of getting pairs with such a difference is quite low.

Another improvement (which was also mentioned in the original publication on differential cryptanalysis) is using an extended form of differences, in which not all the bits of the difference are fixed. This type of differences was later called truncated differences [10]. An important type of truncated differences (in most cases truncated differences refer to this type) is the word-wise truncated differences. Word-wise truncated differences are differences in which the difference itself is not considered, but instead the differences are divided into two classes, namely zero differences and non-zero differences. In these cases the data blocks are divided to words (either 8-bit bytes, or 16-bit words, or words of a different size depending on the native structure of the cipher), and the analysis only considers whether the difference of a word is expected to be zero or not. Such consideration is useful when non-zero differences evolve to other (unknown in advance) non-zero differences, so that the information on the zero/non-zero difference evolve through many rounds of the cipher.

A third extension defines non-XOR differences, such as subtraction of integers (useful for cases where the native operation in the cipher is addition), or differences of division modulo a prime (useful for cases where the native operation in the cipher is multiplication modulo a prime, such as in IDEA [11]). Also a combination of different differences for different parts of the block, or for different rounds of the cipher is considered. For such cases, difference distribution tables where the input differences are defined with one operation and the output differences with another, are very useful (especially when the operation natively transforms one operation to another, such as in cases of exponentiation S boxes, or logarithm S boxes).

Higher-order differences [12] consider derivatives of a second or a higher order. Higher-order differences are shown successful in several cases where differential cryptanalysis is not applicable due to low probabilities of characteristics; in some of these cases higher-order differences prove the most successful attack. However, higher-order attacks are successful mainly against ciphers with a small number of rounds.

It was also observed that in most differential attacks, the intermediate differences predicted by the characteristics are not used, and thus can be ignored [11]. In such cases, the considered differences are only the plaintext differences and the difference after the final round of the characteristic. In most cases there are many different characteristics with the same plaintext difference and the same final difference; these characteristics sum up to one differential, whose probability is the sum of their probabilities.

The major method for protection against differential cryptanalysis is by bounding the probability of the best characteristic (or differential) to be very low. Whenever the designer wishes to prove that differential cryptanalysis is not applicable, he bounds the probability p of the best characteristic (or differential) such that $1/p$ is larger than the required complexity, or even larger than the size of the plaintext space (in which case even choosing the whole plaintext space is not sufficient for mounting an attack). These bounds were formalized into various theories of provable security against differential cryptanalysis.

A specially interesting theory for provable security against differential cryptanalysis (and also linear cryptanalysis) is the theory of decorrelation [16], which makes it possible to prove security of block ciphers against certain (restricted) kinds of attacks, including basic variants of differential and linear cryptanalysis.

Although the usual claims for security against differential cryptanalysis say that the probabilities of the highest-probability differentials are very low, and thus differential attacks require a huge amount of data and complexity, it was observed that even differentials with probability zero (i.e., that cannot occur—there are no right pairs under any key) can be used for attacks [1,9]. This kind of attacks is called *differential cryptanalysis using impossible differentials* (or shortly *impossible cryptanalysis*). The main idea is to select a large set of pairs with the plaintext difference of an impossible differential with $n - 1$ (or slightly less) rounds, where n is the number of rounds of the block cipher, and to try all the possible subkeys of the extra round(s). If it appears that for some value of the subkey, decryption of the ciphertexts by one round (or the few rounds) leads to the impossible difference in any one of the pairs, then we are assured that the subkey is wrong, and thus can be discarded. After discarding sufficiently many subkeys, the attacker reduces his list of possible values of the subkeys to a short list (or even to one subkey), and he is assured that the correct subkey is in the list. Depending on the design of the cipher and the key schedule, for some ciphers it would be more efficient to try reducing the number of possible subkeys to 1 (i.e., only the correct subkey), while for others it would be more efficient to reduce the size of the short list to some larger size, and then perform an exhaustive search of the remaining possible keys.

There are also attacks that use differentials as their building blocks, while combining differentials in various ways. The most promising ones are boomerang [17], amplified boomerang [8], and rectangle [2] attacks. The main idea in all these attacks are the combination of four plaintexts, which for simplicity of description we assume are located on the corners of a square, where one short differential is used in both pairs for the first few rounds (the horizontal edges), while a second short differential is used for the rest of the rounds but on the orthogonal pairs (the vertical edges). Although the probabilities of the total structure are p^2q^2 where p and q are the probabilities of the two differentials, it appears that it is much easier in various cases to find good short differentials, than to find one full differential of a comparable probability.

Although differential cryptanalysis is basically a chosen plaintext attack (as the attacker needs to choose the plaintext differences), the attacker usually does not need to choose the exact values of the plaintexts. This observation allows conversion of chosen plaintext differential cryptanalysis attacks into known plaintext attacks [3], using the fact that in a sufficiently large set of random plaintexts there are many pairs whose difference is as required by the chosen plaintext attack. Once these pairs of plaintexts are identified, the original chosen plaintext attack may be performed on these pairs. This variant usually requires a huge number of known plaintexts, which is about $\sqrt{m2^{n+1}}$ where n is the size of the plaintext in bits and m is the number of chosen plaintext pairs required by the chosen plaintext attack. On some ciphers this is the best published known-plaintext attack.

In some cases it is also possible to convert differential cryptanalysis to ciphertext-only attacks. For more information on these conversions see [7].

Differential cryptanalysis was originally developed on FEAL-8 [13,15], a block cipher which was claimed to be faster and more secure than DES. It was then generalized and extended to DES and other schemes. Feal-8 was broken using a few hundred chosen plaintexts. Given the corresponding ciphertexts, it takes less than a minute on a personal computer to recover the key [5]. The first results on DES [4] showed that DES reduced to 15 rounds was vulnerable to a differential attack, while the full 16-round DES required 2^{58} chosen plaintexts for a successful attack, whose generation is slower than exhaustive search. In the following year an improvement of the technique was invented [6]. The main trick in the improved attack was the ability to receive the first round for free, using large specially designed structures, setting the characteristic from the second round on. This improvement made it possible to apply the 15-round attack on the full 16 rounds. Another improvement allowed to find the key when the first right pair is analyzed, rather than to wait till sufficiently many right pairs are found. This improvement is applicable when the attack considers all the key bits (or almost all the key bits) in a single counting phase. As a result, the improved attack could analyze the full 16-round DES given 2^{47} chosen plaintext and their corresponding ciphertexts, whose complexity of analysis was smaller than 2^{40}.

Eli Biham

References

[1] Biham, Eli, Alex Biryukov, and Adi Shamir (1999). "Cryptanalysis of skipjack reduced to 31 rounds using impossible differentials." *Advances in Cryptology—EUROCRYPT'99*, Lecture Notes in Computer Science, vol. 1592, ed. J. Stern. Springer, Berlin, 12–23.

[2] Biham, Eli, Orr Dunkelman, and Nathan Keller (2002). "New results on boomerang and rectangle attacks." *Proceedings of Fast Software Encryption*, Leuven, Lecture Notes in Computer Science, vol. 2365, eds. Daemen, J. and V. Rijmen. Springer, Berlin, 1–16.

[3] Biham, Eli and Adi Shamir (1993). *Differential Cryptanalysis of the Data Encryption Standard.* Springer-Verlag, Berlin, New York.

[4] Biham, Eli and Adi Shamir (1991). "Differential cryptanalysis of DES-like cryptosystems." *Journal of Cryptology*, 4 (1), 3–72.

[5] Biham, Eli and Adi Shamir (1991). "Differential cryptanalysis of FEAL and N-hash." Technical report CS91-17, Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, *Advances in Cryptology—EUROCRYPT'91*. The extended abstract appears in Lecture Notes in Computer Science, vol. 547, ed. D.W. Davies. Springer, Berlin, 1–16.

[6] Biham, Eli and Adi Shamir (1992). "Differential cryptanalysis of the full 16-round DES." *Advances in Cryptology—CRYPTO'92*, Lecture Notes in Computer Science, vol. 740, ed. E.F Brickel. Springer, Berlin, 487–496.

[7] Biryukov, Alex and Eyal Kushilevitz (1998). "From differential cryptanalysis to ciphertext-only attacks." *Advances in Cryptology—CRYPTO'98*, Lecture Notes in Computer Science, vol. 1462, ed. H. Krawczyk. Springer, Berlin, 72–88.

[8] Kelsey, John, Tadayoshi Kohno, and Bruce Schneier (2000). "Amplified boomerang attacks against reduced-round MARS and serpent." *Proceedings of Fast Software Encryption 7*, Lecture Notes in Computer Science, vol. 1978, ed. B. Schneier. Springer-Verlag, Berlin, 75–93.

[9] Knudsen, Lars Ramkilde (1998). "DEAL—a 128-bit block cipher." AES submission, available on http://www.ii.uib.no/~larsr/papers/deal.ps

[10] Knudsen, Lars (1995). "Truncated and higher order differentials." *Proceedings of Fast Software Encryption 2*, Lecture Notes in Computer Science, vol. 1008, ed. B. Preneel. Springer-Verlag, Berlin, 196–211.

[11] Lai, Xuejia, James L. Massey, and Sean Murphy (1991). "Markov ciphers and differential cryptanalysis." *Advances in Cryptology, Proceedings of EUROCRYPT'91*, Lecture Notes in Computer Science, vol. 547, ed. D.W. Davies. Springer, Berlin, 17–38.

[12] Lai, Xuejia (1994). "Higher order derivative and differential cryptanalysis." *Proceedings of Symposium on Communication, Coding and Cryptography*, in honor of J.L. Massey on the occasion of his 60th birthday.

[13] Miyaguchi, Shoji, Akira Shiraishi, and Akihiro Shimizu (1988). "Fast data encryption algorithm FEAL-8." *Review of Electrical Communications Laboratories*, 36 (4), 433–437.

[14] National Bureau of Standards (1977), Data Encryption Standard, U.S. Department of Commerce, FIPS pub. 46.

[15] Shimizu, Akihiro and Shoji Miyaguchi (1987). "Fast data encryption algorithm FEAL." *Advances in Cryptology—EUROCRYPT'87*, Lecture Notes in Computer Science, vol. 304, eds. David Chaum and Wyn L. Price. Springer, Berlin, 267–278.

[16] Vaudenay, Serge (1998). "Provable security for block ciphers by decorrelation." *Proceedings of STACS'98*, Lecture Notes in Computer Science, vol. 1373, eds. M. Morvan, C. Meinel, and D. Krob. Springer, Berlin, 249–275.

[17] Wagner, David (1999). "The boomerang attack." *Proceedings of Fast Software Encryption, FSE'99, Rome*, Lecture Notes in Computer Science, vol. 1636, ed. L. Knudsen. Springer, Berlin, 156–170.

DIFFERENTIAL–LINEAR ATTACK

Differential–Linear attack is a chosen plaintext *two-stage* technique of cryptanalysis (by analogy with two-stage rocket technology) in which the first stage is covered by differential cryptanalysis, which ensures propagation of useful properties midway through the block cipher. The second stage is then performed from the middle of the cipher and to the ciphertext using linear cryptanalysis. The technique was discovered and demonstrated on the example of 8-round *DES* (see Data Encryption Standard) by Langford and Hellman [4]. Given a *differential characteristic* with probability p for the rounds $1, \dots, i$ and the *linear characteristic* with bias q for the rounds $i + 1, \dots, R$, the bias of resulting linear approximation would be $1/2 + 2pq^2$ and the data complexity of the attack will be $O(p^{-2}q^{-4})$ [3, p. 65]. Thus the attack would be useful only in special cases when there are good characteristics or linear approximations half-way through the cipher, but no good patterns for the full cipher. Their attack enhanced with such refinements as packing data into *structures* and *key-ranking* (or *list decoding*) can recover 10-bits of the secret key for 8-round DES using 512 chosen plaintexts. In [1] the same technique is used to break 8-round FEAL with 12 chosen plaintexts and expensive analysis phase. Further applications and refinements of the technique are given in [2].

Alex Biryukov

References

[1] Aoki, K. and K. Ohta (1996). "Differential-linear cryptanalysis of FEAL-8." *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, E79A (1), 20–27.

[2] Eli Biham, Orr Dunkelman, and Nathan Keller, "Enhancing Differential-Linear Cryptanalysis", *Advances in Cryptology ASIACRYPT 2002*, Lecture Notes in Computer Science, vol. 2501, ed. Y. Zheng. Springer-Verlag, Berlin, p. 254–266.

[3] Langford, S.K. (1995). "Differential-linear cryptanalysis and threshold signatures." Technical report, *PhD Thesis*, Stanford University.

[4] Langford, S.K. and M.E. Hellman (1994). "Differential–linear cryptanalysis." *Advances in Cryptology—CRYPTO'94*. Lecture Notes in Computer Science, vol. 839, ed. Y. Desmedt. Springer-Verlag, Berlin, 17–25.

DIFFERENTIAL POWER ANALYSIS

Differential Power Analysis utilizes power consumption of a cryptographic device such as a smartcard as side-channel information. In *Simple Power Analysis* (SPA) an attacker directly observes a device's power consumption. It is known that the amount of power consumed by the device varies depending on the data operated on and the instructions performed during different parts of an algorithm's execution. Define a *power trace* as a set of power consumption measurements during a cryptographic operation. By simply examining power traces, it is possible to determine major characteristic details of a cryptographic device and the implementation of the cryptographic algorithm being used. SPA can therefore be used to discover implementation details, such as *DES* rounds (see Data Encryption Standard) and *RSA* operations (see RSA public key encryption). Moreover, SPA can reveal differences between multiplication and squaring operations, which can be used to recover the private key in RSA implementations. SPA can also reveal visible differences within permutations and shifts in DES implementations, which might lead to recovering the secret DES key.

While SPA attacks use primarily visual inspection to identify relevant power fluctuations, Differential Power Analysis (DPA) exploits characteristic behavior (e.g., power consumption behavior of transistors and logic gates) [2]. DPA uses an attacking model and statistical analysis to extract hidden information from a large sample of power traces obtained during "controlled" cryptographic computations. In case of SPA, direct observations of a device's power consumption would not allow identifying the effects of a single transistor switching. The use of statistical methods in a controlled DPA environment allows identifying small differences in power consumption, which can be used to recover specific information such as the individual bits in a secret key. This means secret key material can be recovered from tamper-resistant devices such as smartcards (smartcard tamper resistance). To execute an attack based on DPA, an attacker does not need to know as many details about how the algorithm is implemented.

The basis of a DPA attack is the use of an abstract model based on the power consumption characteristics of the logic that includes the noise components. When measuring the power consumption, various noise components are superimposed on the power traces. The main noise sources are external, intrinsic, quantization and algorithmic noise. Intrinsic and quantization noise are small compared to the power consumption. The external noise can be reduced by careful use of the measurement equipment. The algorithmic noise can be averaged out by the DPA strategy itself. To reduce the influence of noise in DPA one can increase the number of samples required to detect variations. Analysis can take place in the time and frequency domain.

The basis DPA technique is as follows. Assume that a sufficient number N of random power traces have been collected (e.g., N samples of ciphertexts obtained using the same encryption key). Each power trace is a collection of power samples $\mathrm{PS}(n, t)$, which represent the power consumption at time t in trace n as the sum of the power dissipated by all circuitry. In practice, the number of measurements t in each power trace depends on the sampling rate and the memory capacity as well as the duration of the cryptographic operation. Next, partition the power samples $\mathrm{PS}(n, t)$ into two sets S_0 and S_1 according to the outcome 0 or 1 of a partitioning or discrimination function D. The outcome value of the partitioning function D can be simply the value of a specific ciphertext bit. In general, the size of set S_0 will be roughly the same as the size of S_1. Next, compute the average power signal for each set S at time t. By subtracting the two averages, we obtain the DPA bias signal $B(t)$. Selecting an appropriate D-function will result in a DPA bias signal that an attacker can use to verify guesses of the secret key. The D-function is chosen such that at some point during implementation the device needs to calculate the value of this bit. When this occurs or any time data containing this bit is manipulated, there will be a slight difference in the amount of power dissipated depending on whether this bit is a zero or a one. Let ϵ denote this difference, and the instruction manipulating the D-bit occurs at time t', then the value ϵ is equal to the expectation difference

$$E[S \mid (D = 0)] - E[S \mid (D = 1)], \qquad \text{for } t = t'.$$

When $t \neq t'$ the device is manipulating bits other than the D-bit, and assuming that the power dissipation is independent of the D-bit, the difference in expectation of the two sets equals zero for sufficiently large N. Thus the bias function $B(t)$ will show power spikes of height ϵ at times t' and will appear flat at all other times. If the proper D-function was chosen, the bias signal will show spikes whenever the D-bit was manipulated and otherwise the resulting $B(t)$ will not show any bias. Using this approach an attacker can verify guesses for the hidden key bit information using the D-function. Repeating this approach for different D-bits, the secret key can be obtained bit by bit.

Variants or improvements of the classical DPA attack exist that use signals from multiple sources, use different measuring techniques, combine signals with different temporal offsets, use specific and more powerful differential functions, and apply more advanced signal processing functions and models. To enlarge the peak, a multiple-bit attack can be used.

A DPA attack involves hundreds to thousands of samples. After processing and statistical analysis, the DPA process can reconstruct the full secret or private key within several minutes. The whole process is easy to implement and requires only standard measurement equipment, which cost lies between a few hundred to a few thousand dollars. DPA attacks are non-invasive, which makes them difficult to detect. DPA requires little or no information about the target device and can be automated. DPA and SPA has successfully been applied to attack a large number of smartcards and PCMCIA cards [3]. See [1] for an approach how to counteract Power Analysis attacks.

Tom Caddy

References

[1] Chari, Suresh, Charanjit Jutla, Josyula R. Rao, and Pankaj Rohatgi (1999). "Towards sound approaches to counteracts power-analysis attacks." *Advances in Cryptology—CRYPTO'99*, Lecture Notes in Computer Science, vol. 1666, ed. M. Wiener. Springer-Verlag, Berlin, 389–412.

[2] Kocher, Paul, Joshua Jaffe, and Benjamin Jun (1999). "Differential power analysis." *Advances in Cryptology—CRYPTO'99*, Lecture Notes in Computer Science, vol. 1666, ed. M. Wiener. Springer-Verlag, Berlin, 388–397.

[3] Messerges, Thomas S., Ezzy A. Dabbish, and Robert H. Sloan (1999). "Investigations of power analysis attacks on smartcards." *Proceedings of USENIX Workshop on Smartcard Technology*, 151–161.

DIFFIE–HELLMAN KEY AGREEMENT

The Diffie–Hellman protocol is a type of key agreement protocol. It was originally described in Diffie and Hellman's seminal paper on public key cryptography.

This key agreement protocol allows Alice and Bob to exchange public key values, and from these values and knowledge of their own corresponding private keys, securely compute a shared key K, allowing for further secure communication. Knowing only the exchanged public key values, an eavesdropper is not able to compute the shared key.

As a preamble to the protocol, the following public parameters are assumed to exist (see Number Theory): a large prime number p such that discrete logarithms in the multiplicative group of integers from 1 to $p-1$ (Z_p^*) are intractable; and a generator g of Z_p^*. Alice randomly selects a value $0 < \mathrm{a} < p-1$ and computes $r = g^a \bmod p$. Alice sends r to Bob. Similarly, Bob selects a value $0 < \mathrm{b} < p-1$ and computes $s = g^b \bmod p$. Bob sends s to Alice. Given a and s, Alice computes $K = s^a \bmod p \equiv g^{ab} \pmod p$. Similarly, given b and r, Bob computes $K = r^b \bmod p \equiv g^{ab} \pmod p$. Thus, Alice and Bob are able to compute the same key value, K.

Now consider the information available to an eavesdropper. This includes g, p, r and s. Thus, the eavesdropper must attempt to compute $K \equiv g^{ab} \pmod p$ given $g^a \bmod p$ and $g^b \bmod p$. This is known as the decisional Diffie–Hellman *problem* and for appropriately chosen g and p, it is believed to be very difficult to solve.

Several variations to this simple protocol exist (see Key Agreement). Of particular note is the fact that the above protocol does not provide for the authentication of Alice and Bob. The Station-to-Station protocol provides one variation to this protocol that authenticates Alice and Bob.

Mike Just

References

[1] Menezes, Alfred, Paul van Oorschot, and Scott Vanstone (1997). *Handbook of Applied Cryptography*. CRC Press, Boca Raton.

[2] Stinson, Douglas R. (1995). *Cryptography: Theory and Practice*. CRC Press, Boca Raton.

DIFFIE–HELLMAN PROBLEM

In their pioneering paper Diffie and Hellman [15] proposed an elegant, reliable, and efficient way to establish a common key between two communicating parties. In the most general settings their idea can be described as follows (see Diffie–Hellman key agreement for further discussion). Given a cyclic group $\mathcal{G}$ and a generator g of $\mathcal{G}$, two communicating parties Alice and Bob execute the following protocol:

- Alice selects *secret* x, Bob selects *secret* y;
- Alice publishes $X = g^x$, Bob publishes $Y = g^y$;
- Alice computes $K = Y^x$, Bob computes $K = X^y$.

Thus at the end of the protocol the values $X = g^x$ and $Y = g^y$ have become *public*, while the value $K = Y^x = X^y = g^{xy}$ supposedly remains *private* and is known as the *Diffie–Hellman secret key*.

Thus the *Diffie–Hellman Problem*, DHP, with respect to the group $\mathcal{G}$ is to compute g^{xy} from the given values of g^x and g^y.

Certainly, only groups in which DHP is hard are of cryptographic interest. For example, if $\mathcal{G}$ is an *additive* group of the residue ring $\mathbb{Z}_m$ modulo m, see modular arithmetic, then DHP is trivial: using additive notations the attacker simply computes $x \equiv X/g \pmod m$ (because g is a generator of the additive group of $\mathbb{Z}_m$, we have $\gcd(g, m) = 1$) and then $K \equiv xY \pmod m$.

On the other hand, it is widely believed that using *multiplicative* subgroups of the group of units $\mathbb{Z}_m^*$ of the residue ring $\mathbb{Z}_m$ modulo m yields examples of groups for which DHP is hard, provided that the modulus m is carefully chosen. This belief also extends to subgroups of the multiplicative group $\mathbb{F}_q^*$ of a *finite field* $\mathbb{F}_q$ of q elements. In fact these groups are exactly the groups suggested by Diffie and Hellman [15]. Although, since that time the requirements on the suitable groups have been refined and better understood, unfortunately not too many other examples of "reliable" groups have been found. Probably the most intriguing and promising example, practically and theoretically, is given by subgroups of point groups on elliptic curves, which have been proposed for this kind of application by Koblitz [24] and Miller [36]. Since the original proposal, many very important theoretical and practical issues related to using elliptic curves in cryptography have been investigated, see [2, 17]. Even more surprisingly, elliptic curves have led to certain variants of the Diffie–Hellman schemes, which are not available in subgroups of $\mathbb{F}_q^*$ or $\mathbb{Z}_m^*$, see [5, 22, 23] and references therein.

Diffie–Hellman and Discrete Logarithm Problems: It is immediate that if one can find x from the given value of $X = g^x$, that is, solve the discrete logarithm problem, DLP, then the whole scheme is broken. In fact, in our example of a "weak" group $\mathcal{G}$, this is exactly DLP which can easily be solved. Thus DHP is not harder than DLP. On the other hand, the only known (theoretical and practical) way to solve DHP is to solve the associated DLP. Thus a natural question arises whether DHP is equivalent to DLP or is strictly weaker. The answer can certainly depend on the specific group $\mathcal{G}$.

Despite a widespread assumption that this indeed is the case, that is, that in any cryptographically "interesting" group DHP and DLP are equivalent, very few theoretical results are known. In particular, it has been demonstrated in [6, 31, 32] that, under certain conditions, DHP and DLP are polynomial time equivalent. However, there are no unconditional results known in this direction.

Some quantitative relations between complexities of DHP and DLP are considered in [13].

Cryptographically Interesting Groups: As we have mentioned, the choice of the group $\mathcal{G}$ is crucial for the hardness of DHP (while the choice of the generator g does not seem to be important at all). Probably the most immediate choice is $\mathcal{G} = \mathbb{F}_q^*$, thus g is a primitive element of $\mathbb{F}_q$. However, one can work in a subgroup of $\mathbb{F}_q^*$ of sufficiently large prime order ℓ (but still much smaller than q and thus more efficient) without sacrificing the security of the protocol. Indeed, we recall that based on our current knowledge we may conclude that the hardness of DLP in a subgroup $\mathcal{G} \subseteq \mathbb{F}_q^*$ (at least for some most commonly used types of fields; for further discussion see discrete logarithm problem) is majorised

1. by $\ell^{1/2}$ where ℓ is the largest prime divisor of $\#\mathcal{G}$, see [35, 44];
2. by $L_q\,[1/2, 2^{1/2}]$ for a rigorous unconditional algorithm, see [37];
3. by $L_q\left[1/3, (64/9)^{1/3}\right]$ for the heuristic number field sieve algorithm, see [39, 40],

where as usual we denote by $L_x[t, \gamma]$ (see L-notation) any quantity of the form

$$L_x[t, \gamma] = \exp((\gamma + o(1))(\log x)^t(\log\log x)^{1-t}).$$

It has also been discovered that some special subgroups of some special extension fields are computationally more efficient and also allow one to reduce the information exchange without sacrificing the security of the protocol. The two most practically and theoretically important examples are given by LUC, see [3,43], and XTR, see [26–28], protocols (see, more generally, subgroup cryptosystems). Despite several substantial achievements in this area, these results are still to be better understood and put in a more systematic form [10].

One can also consider subgroups of the residue ring $\mathbb{Z}_m^*$ modulo a composite $m \geq 1$. Although they do not seem to give any practical advantages (at least in the original setting of the two party key exchange protocol), there are some theoretical results supporting this choice, for example, see [1].

The situation is more complicated with subgroups of the point groups of elliptic curves, and more generally of abelian varieties. For these groups not only the arithmetic structure of the cardinality $\mathcal{G}$ matters, but many other factors also play an essential role, see [2, 17, 19, 20, 25, 34, 38] and references therein.

Bit Security of the Diffie–Hellman Secret Key: So far, while there are several examples of groups in which DHP (like DLP) is conjectured to be hard, as with other areas of cryptography, the security relies on unproven assumptions. Nevertheless, after decades of exploration, we have gained a reasonably high level of confidence in some groups, for example, in subgroups of $\mathbb{F}_p^*$. Of course, this assumes that p and $\#\mathcal{G}$ are sufficiently large to thwart the discrete logarithm attack. Typically, nowadays, p is at least about 1024 bits, $\#\mathcal{G}$ is at least about 160 bits. However, after the common key $K = g^{xy}$ is established, only a small portion of bits of K will be used as a common key for some pre-agreed symmetric cryptosystem.

Thus, a natural question arises: *Assume that finding all of K is infeasible, is it necessarilly infeasible to find certain bits of K?*

In practice, one often derives the secret key from K via a hash function but this requires an additional function, which generally must be modeled as a black box. Moreover, this approach requires a hash function satisfying some additional requirements which could be hard to prove unconditionally. Thus the security of the the obtained private key relies on the hardness of DHP and some assumptions about the hash function. Bit security results allow us to eliminate the usage of hash functions and thus to avoid the need to make any additional assumptions.

For $\mathcal{G} = \mathbb{F}_p^*$, Boneh and Venkatesan [8] have found a very elegant way, *using lattice basis reduction* (see lattices), to solve this question in the affirmative, see also [9]. Their result has been slightly improved and also extended to other groups in [21]. For the XTR version of DHP it has recently been done in [30]. The results of these papers can be summarized as follows: "error-free" recovery of even some small portion of information about the Diffie–Hellman secret key $K = g^{xy}$ is as hard as recovering the whole key (cf. hard-core bit). Including the case where the recovering algorithm works with only some non-negligible positive probability of success is an extremely important open question. This would immediately imply that hashing K does not increase the security of the secret key over simply using a short substring of bits of K for the same purpose, at least in an asymptotic sense.

It is important to remark that these results do not assert that the knowledge of just a few bits of K for *particular* (g^x, g^y) translates into the knowledge of all the bits. Rather the statement is that given an efficient algorithm to determine certain bits of the common key corresponding to *arbitrary* g^x and g^y, one can determine all of the common key corresponding to *particular* g^x and g^y.

Another, somewhat dual problem involving some partial information about K is studied in [41]. It is shown in [41] that any polynomial time algorithm which for given x and y produces a list $\mathcal{L}$ of polynomially many elements of $\#\mathcal{G}$ where $K = g^{xy} \in \mathcal{L}$, can be used to design a polynomial time algorithm which finds K unambiguously.

NUMBER THEORETIC AND ALGEBRAIC PROPERTIES: As we have mentioned, getting rigorous results about the hardness of DHP is probably infeasible nowadays. One can however study some number theoretic and algebraic properties of the map $K : \mathcal{G} \times \mathcal{G} \to \mathcal{G}$ given by $K(g^x, g^y) = g^{xy}$. This is of independent intrinsic interest and may also shed some light on other properties of this map which are of direct cryptographic interest.

For example, many cryptographic protocols are based on the assumption of hardness of the decisional Diffie–Hellman problem, DDHP, rather than DHP itself. Roughly speaking, DDHP is the problem of deciding whether a triple $(u, v, w) \in \mathcal{G}^3$ of random elements of $\mathcal{G}$ is of the form (g^x, g^y, g^{xy}) for some x and y. Clearly, DDHP is no harder than DHP, and it is believed that in fact it is no easier, see [4]. Unfortunately there are no viable approaches to a proof of this conjecture. Motivated by this problem, in the series of works [11,12,18] several "statistical" results have been established, which show that if $\mathcal{G}$ is a sufficiently large subgroup of $\mathbb{F}_p^*$ then at least statistically the triples (g^x, g^y, g^{xy}) behave as triples of random elements.

One can also study algebraic properties of the set of points (g^x, g^y, g^{xy}) or even just (g^x, g^{x^2}) (which corresponds to the "diagonal" case $x = y$). In particular one can ask about the degree of polynomials F for which $F(g^x, g^y, g^{xy}) = 0$ or $F(g^x, g^y) = g^{xy}$ or $F(g^x, g^{x^2}) = 0$ or $F(g^x) = g^{x^2}$ for all or "many" $x, y \in \mathcal{G}$. Certainly it is intuitively obvious that such polynomials should be of very large degree and have a complicated structure. It is useful to recall the interpolation attack on block ciphers which is based on finding polynomial relations of similar spirit. It has been shown in [14] (as one would certainly expect) that such polynomials are of exponentially large degree, see also [42]. Several more results of this type can also be found in [16,33,45].

Igor E. Shparlinski

References

[1] Biham, E., D. Boneh, and O. Reingold (1999). "Breaking generalized Diffie–Hellman modulo a composite is no easier than factoring." *Inform. Proc. Letters*, 70, 83–87.

[2] Blake, I., G. Seroussi, and N. Smart (1999). *Elliptic Curves in Cryptography*. London Mathematical Society, Lecture Notes Series, vol. 265. Cambridge University Press, Cambridge.

[3] Bleichenbacher, D., W. Bosma, and A.K. Lenstra (1995). "Some remarks on Lucas-based cryptosystems." *Advances in Cryptology—CRYPTO'95*, Lecture Notes in Computer Science, vol. 963, ed. D. Coppersmith. Springer-Verlag, Berlin, 386–396.

[4] Boneh, D. (1998). "The decision Diffie–Hellman problem." *Proceedings of ANTS-III*, Lecture Notes in Computer Science, vol. 1423, ed. J.P. Buhler. Springer-Verlag, Berlin, 48–63.

[5] Boneh, D. and M. Franklin (2001). "Identity-based encryption from the Weil pairing." *Advances in Cryptology—CRYPTO 2001*, Lecture Notes in Computer Science, vol. 2139, ed. J. Kilian. Springer-Verlag, Berlin, 213–229.

[6] Boneh, D. and R. Lipton (1996). "Algorithms for black-box fields and their applications to cryptography." *Advances in Cryptology—CRYPTO'96*, Lecture Notes in Computer Science, vol. 1109, ed. N. Koblitz. Springer-Verlag, Berlin, 283–297.

[7] Boneh, D. and I.E. Shparlinski (2001). "On the unpredictability of bits of the elliptic curve Diffie–Hellman scheme." *Advances in Cryptology—CRYPTO 2001*, Lecture Notes in Computer Science, vol. 2139, ed. J. Kilian. Springer-Verlag, Berlin, 201–212.

[8] Boneh, D. and R. Venkatesan (1996). "Hardness of computing the most significant bits of secret keys in Diffie–Hellman and related schemes." *Advances in Cryptology—CRYPTO'96*, Lecture Notes

in Computer Science, vol. 1109, ed. N. Koblitz. Springer-Verlag, Berlin, 129–142.

[9] Boneh, D. and R. Venkatesan (1997). "Rounding in lattices and its cryptographic applications." *Proc. 8th Annual ACM-SIAM Symp. on Discr. Algorithms*. ACM, 675–681.

[10] Bosma, W., J. Hutton, and E. Verheul (2002). "Looking beyond XTR." *Advances in Cryptography—ASIACRYPT 2002*, Lecture Notes in Computer Science, vol. 2501, ed. Y. Zheng. Springer-Verlag, Berlin, 46–63.

[11] Canetti, R., J.B. Friedlander, S.V. Konyagin, M. Larsen, D. Lieman, and I.E. Shparlinski (2000). "On the statistical properties of Diffie–Hellman distributions." *Israel J. Math.*, 120, 23–46.

[12] Canetti, R., J.B. Friedlander, and I.E. Shparlinski (1999). "On certain exponential sums and the distribution of Diffie–Hellman triples." *J. London Math. Soc.*, 59, 799–812.

[13] Cherepnev, M.A. (1996). "On the connection between the discrete logarithms and the Diffie–Hellman problem." *Diskretnaja Matem.*, 6, 341–349 (in Russian).

[14] Coppersmith, D. and I.E. Shparlinski (2000). "On polynomial approximation of the discrete logarithm and the Diffie–Hellman mapping." *J. Cryptology*, 13, 339–360.

[15] Diffie, W. and M.E. Hellman (1976). "New directions in cryptography." *IEEE Trans. Inform. Theory*, 22, 109–112.

[16] El Mahassni, E. and I.E. Shparlinski (2001). "Polynomial representations of the Diffie–Hellman mapping." *Bull. Aust. Math. Soc.*, 63, 467–473.

[17] Enge, A. (1999). *Elliptic Curves and their Applications to Cryptography*. Kluwer Academic Publishers, Dordrecht.

[18] Friedlander, J.B. and I.E. Shparlinski (2001). "On the distribution of Diffie–Hellman triples with sparse exponents." *SIAM J. Discr. Math.*, 14, 162–169.

[19] Galbraith, S.D. (2001). "Supersingular curves in cryptography." *Advances in Cryptology—ASIACRYPT 2001*, Lecture Notes in Computer Science, vol. 2248, ed. C. Boyd. Springer-Verlag, Berlin, 495–513.

[20] Gaudry, P., F. Hess, and N.P. Smart (2002). "Constructive and destructive facets of Weil descent on elliptic curves." *J. Cryptology*, 15, 19–46.

[21] González Vasco, M.I. and I.E. Shparlinski (2001). "On the security of Diffie–Hellman bits." *Proc. Workshop on Cryptography and Computational Number Theory, Singapore, 1999*. Birkhäuser, 257–268.

[22] Joux, A. (2000). "A one round protocol for tripartite Diffie–Hellman." *Proc. of ANTS-IV*, Lecture Notes in Computer Science, vol. 1838, ed. W. Bosma. Springer-Verlag, Berlin, 385–393.

[23] Joux, A. (2002). "The Weil and Tate pairings as building blocks for public key cryptosystems." *Proc. of ANTS V*, Lecture Notes in Computer Science, vol. 2369, eds. D. Kohel and C. Fieker. Springer-Verlag, Berlin, 20–32.

[24] Koblitz, N. (1987). "Elliptic curve cryptosystems." *Math. Comp.*, 48, 203–209.

[25] Koblitz, N. "Good and bad uses of elliptic curves in cryptography." *Moscow Math. Journal*. To appear.

[26] Lenstra, A.K. and E.R. Verheul (2000). "The XTR public key system." *Advances in Cryptology—CRYPTO 2000*, Lecture Notes in Computer Science, vol. 1880, ed. M. Bellare. Springer-Verlag, Berlin, 1–19.

[27] Lenstra, A.K. and E.R. Verheul (2000). "Key improvements to XTR." *Advances in Cryptography—ASIACRYPT 2000*, Lecture Notes in Computer Science, vol. 1976, ed. T. Okamoto. Springer-Verlag, Berlin, 220–233.

[28] Lenstra, A.K. and E.R. Verheul (2001). "Fast irreducibility and subgroup membership testing in XTR." *PKC 2001*, Lecture Notes in Computer Science, vol. 1992, ed. K. Kim. Springer-Verlag, Berlin, 73–86.

[29] Lenstra, A.K. and E.R. Verheul (2001). "An overview of the XTR public key system." *Proc. the Conf. on Public Key Cryptography and Computational Number Theory, Warsaw 2000*. Walter de Gruyter, 151–180.

[30] Li, W.-C.W., M. Näslund, and I.E. Shparlinski (2002). "The hidden number problem with the trace and bit security of XTR and LUC." *Advances in Cryptology—CRYPTO 2002*, Lecture Notes in Computer Science, vol. 2442, ed. M. Yung. Springer-Verlag, Berlin, 433–448.

[31] Maurer, U.M. and S. Wolf (1999). "The relationship between breaking the Diffie–Hellman protocol and computing discrete logarithms." *SIAM J. Comp.*, 28, 1689–1721.

[32] Maurer, U.M. and S. Wolf (2000). "The Diffie–Hellman protocol." *Designs, Codes and Cryptography*, 19, 147–171.

[33] Meidl, W. and A. Winterhof (2002). "A polynomial representation of the Diffie–Hellman mapping." *Appl. Algebra in Engin., Commun. and Computing*, 13, 313–318.

[34] Menezes, A.J., N. Koblitz, and S.A. Vanstone (2000). "The state of elliptic curve cryptography." *Designs, Codes and Cryptography*, 19, 173–193.

[35] Menezes, A.J., P.C. van Oorschot, and S.A. Vanstone (1996). *Handbook of Applied Cryptography*. CRC Press, Boca Raton, FL.

[36] Miller, V.C. (1986). "Use of elliptic curves in cryptography." *Advances in Cryptology—CRYPTO'85* Lecture Notes in Computer Science, vol. 218, ed. H.C. Williams. Springer-Verlag, Berlin, 417–426.

[37] Pomerance, C. (1987). "Fast, rigorous factorization and discrete logarithm algorithms." *Discrete Algorithms and Complexity*. Academic Press, 119–143.

[38] Rubin, K. and A. Silverberg (2002). "Supersingular abelian varieties in cryptology." *Advances in Cryptology—CRYPTO 2002*, Lecture Notes in Computer Science, vol. 2442, ed. M. Yung. Springer-Verlag, Berlin, 336–353.

[39] Schirokauer, O. (1993). "Discrete logarithms and local units." *Philos. Trans. Roy. Soc. London, Ser. A*, 345, 409–423.

[40] Schirokauer, O., D. Weber, and T. Denny (1996). "Discrete logarithms: The effectiveness of the index calculus method." *Proceedings of ANTS-II*, Lecture Notes in Computer Science, vol. 1122, ed. H. Cohen. Springer-Verlag, Berlin, 337–362.

[41] Shoup, V. (1997). "Lower bounds for discrete logarithms and related problems." *Advances in Cryptology—EUROCRYPT'97*, Lecture Notes in Computer Science, vol. 1233, ed. W. Fumy. Springer-Verlag, Berlin, 256–266.

[42] Shparlinski, I.E. (2003). *Cryptographic Applications of Analytic Number Theory*. Birkhauser.

[43] Smith, P.J. and C.T. Skinner (1995). "A public-key cryptosystem and a digital signature system based on the Lucas function analogue to discrete logarithms." *Advances in Cryptography—ASIACRYPT'94*, Lecture Notes in Computer Science, vol. 917, eds. J. Pieprzyk and R. Safavi-Naini. Springer-Verlag, Berlin, 357–364.

[44] Stinson, D.R. (1995). *Cryptography: Theory and Practice*. CRC Press, Boca Raton, FL.

[45] Winterhof, A. (2001). "A note on the interpolation of the Diffie–Hellman mapping." *Bull. Aust. Math. Soc.*, 64, 475–477.

DIGITAL SIGNATURE SCHEMES

Digital signature schemes are techniques to assure an entity's acknowledgement of having sent a certain message. Typically, an entity has a private key and a corresponding public key which is tied to the entity's name (see also public key infrastructure). The entity generates a string called *signature* which depends on the message to sign and his private key.

The fact that the entity acknowledged, i.e. that he signed the message, can be verified by anyone using the entity's public key, the message, and the signature. *Data* authentication and signature schemes are sometimes distinguished in the sense that in the latter, verification can be done by anyone at any time after the generation of the signature. Due to this property, the digital signature scheme achieves non-repudiation property, that is, a signer cannot later deny the fact of signing.

Some examples of digital signature schemes are RSA digital signature scheme, ElGamal digital signature scheme, Rabin digital signature scheme, Schnorr digital signature scheme, Digital Signature Standard, and Nyberg-Rueppel signature scheme.

A digital signature scheme consists of three algorithms, namely the *key generation algorithm*, the *signing algorithm* and the *verification algorithm*. The security of digital signature is argued as follows: no adversary, without the knowledge of the private key, can generate a message and a signature that passes the verification algorithm. (See forgery for more discussions on the security of signatures.) There are two types of signature schemes, namely 'with *appendix*' and 'with *message recovery*'. In the former, the target message is the input of the verification algorithm; that is, the verifier must know the message in advance to verify the signature. In the latter, the target message is the output of the verification algorithm, so the message does not need to be sent with the signature. An example of the former is the ElGamal digital signature scheme and of the latter is the RSA digital signature scheme.

Kazue Sako

DIGITAL SIGNATURE STANDARD

The Digital Signature Standard (DSS), first proposed by Kravitz [2] in 1991, became a US federal standard in May 1994. It is published as Federal Information Processing Letters (FIPS) 186. The signature scheme is based on the ElGamal digital signature scheme and borrows ideas from Schnorr digital signatures for reducing signature size. We describe a slight generalization of the algorithm that allows for an arbitrary security parameter, whereas the standard only supports a fixed parameter. The signature scheme makes use of modular arithmetic and works as follows:

Key Generation. Given two security parameters $\tau, \lambda \in \mathbb{Z}$ $(\tau > \lambda)$ as input do the following:

1. Generate a random λ-bit prime q.
2. Generate a random τ-bit prime prime p such that q divides $p - 1$.
3. Pick an element $g \in \mathbb{Z}_p^*$ of order q.
4. Pick a random integer $\alpha \in [1, q]$ and compute $y = g^\alpha \in \mathbb{Z}_p^*$.
5. Let H be a hash function $H : \{0, 1\}^* \to \mathbb{Z}_q$. The FIPS 186 standard mandates that H be based on the SHA-1 cryptographic hash function.
6. Output the public key (p, q, g, y, H) and the private key (p, q, g, α, H).

Signing. To sign a message $m \in \{0, 1\}^*$ using the private key (p, q, g, α, H) do:

1. Pick a random integer $k \in [1, q-1]$.
2. Compute $r = (g^k \bmod p) \bmod q$. We view r as an integer $0 \leq r < q$.
3. Compute $s = k^{-1}(H(m) + \alpha r) \bmod q$.
4. Output the pair $(r, s) \in \mathbb{Z}_p^*$ as the signature on m.

Verifying. To verify a message/signature pair $(m, (r, s))$ using the public key (p, q, g, y, H) do:

1. Verify that $0 \leq r, s < q$, otherwise reject the signature.
2. Compute $u_1 = H(m)/s \bmod q$ and $u_2 = r/s \bmod q$.
3. Compute $v = (g^{u_1} y^{u_2} \bmod p) \bmod q$.
4. Accept the signature if $r = v \bmod q$. Otherwise, reject.

We first check that the verification algorithm accepts all valid message/signature pairs. For a valid message/signature pair we have

$$g^{u_1} y^{u_2} = g^{u_1 + \alpha u_2} = g^{(H(m)+\alpha r)/s} = g^k \pmod{p}.$$

It follows that $v = (g^{u_1} y^{u_2} \bmod p) \bmod q = r$ and therefore a valid message/signature is always accepted.

It is not clear how to analyze the security of this algorithm. Even the random oracle model does not seem to help since there is no hash function in the algorithm that can be modelled as a random oracle. It is believed that this is deliberate so that the algorithm does not infringe on existing patents. Security analysis for a generalization of DSS is given in [1].

To discuss signature length we fix concrete security parameters. At the present time the discrete-logarithm problem in the cyclic group $\mathbb{Z}_p^*$ where p is a 1024-bit prime and is considered intractable [3] except for a very well funded organization. DSS uses a subgroup of order q of $\mathbb{Z}_p^*$. When q is a 160-bit prime, the discrete log problem in this subgroup is believed to be as hard as discrete-log in all of $\mathbb{Z}_p^*$. Hence, for the present discussion we assume p is a 1024-bit prime and q is a 160-bit prime. Since a DSS signature contains two elements in $\mathbb{Z}_q$ we see that, with these parameters, a DSS signature is 320-bits long. This is the same length as a Schnorr signature. We note that BLS short signatures are half the size and provide comparable security.

Dan Boneh

References

[1] Brickell, Ernest, David Pointcheval, Serge Vaudenay, and Moti Yung (2000). "Design validations for discrete logarithm based signature schemes." *Proceedings of Public Key Cryptography 2000*, 276–292.

[2] Kravitz, D. (1993). "Digital signature algorithm." U.S. patent #5,231,668.

[3] Lenstra, Arjen and Eric Verheul (2001). "Selecting cryptographic key sizes." *Journal of Cryptology*, 14 (4), 255–293.

DIGITAL STEGANOGRAPHY

INTRODUCTION: Steganography is the art and science of hiding information by embedding messages within other, seemingly harmless messages. Steganography means "covered writing" in Greek. As the goal of steganography is to hide the *presence* of a message and to create a covert channel, it can be seen as the complement of cryptography, whose goal is to hide the *content* of a message.

A famous illustration of steganography is Simmons' "Prisoners' Problem" [10]: Alice and Bob are in jail, locked up in separate cells far apart from each other, and wish to devise an escape plan. They are allowed to communicate by means of sending messages via trusted couriers, provided they do not deal with escape plans. But the couriers are agents of the warden Eve (who plays the role of the adversary here) and will leak all communication to her. If Eve detects any sign of conspiracy, she will thwart the escape plans by transferring both prisoners to high-security cells from which nobody has ever escaped. Alice and Bob are well aware of these facts, so that before getting locked up, they have shared a secret codeword that they are now going to exploit for embedding a hidden information into their seemingly innocent messages. Alice and Bob succeed if they can exchange information allowing them to coordinate their escape and Eve does not become suspicious.

According to the standard terminology of information hiding [8], a legitimate communication among the prisoners is called *covertext*, and a message with embedded hidden information is called *stegotext*. The distributions of covertext and stegotext are known to the warden Eve because she knows what constitutes a legitimate communication among prisoners and which tricks they apply to add a hidden meaning to innocent looking messages.

The algorithms for creating stegotext with an embedded message by Alice and for decoding the message by Bob are collectively called a *stegosystem*. A stegosystem should hide the embedded message at least as well as an encryption scheme since it may be enough for the adversary to learn only a small amount of information about the embedded message to conclude that Alice and Bob are conspiring. But steganography requires more than that. The ciphertext generated by most encryption schemes resembles a sequence of random bits, and this is very likely to raise the suspicion of Eve. Instead, stegotext should "look" just like innocent covertext even though it contains a hidden message.

This intuition forms the basis of the recently developed formal approach to steganography [2,3,5, 6,11]. It views a stegosystem as a cryptosystem with the additional property that its output, i.e., the stegotext, is not distinguishable from covertext to the adversary.

Formally, a stegosystem consists of a triple of algorithms for key generation, message encoding, and message decoding, respectively. In the symmetric-key setting considered here, the output of the key generation algorithm is given only to Alice and to Bob.

The covertext is modeled by a distribution $\mathcal{C}$ over a given set C. The covertext may be given explicitly as a list of values or implicitly as an oracle that returns a sample of $\mathcal{C}$ upon request. A stegosystem that does not require explicit knowledge of the covertext distribution is called *universal*.

A more general model of a covertext *channel* has also been proposed in the literature [5], which allows to model dependencies among repeated uses of the same covertext source. A channel consists of an unbounded sequence of values drawn from a set C whose distribution may depend in arbitrary ways on past outputs; access to the channel is given only by an oracle that samples from the channel. The assumption is that the channel oracle can be queried with an arbitrary prefix of a possible channel output, i.e., its past "history," and it will return the next symbol according to the channel distribution. In order to simplify the presentation, channels are not considered further here, but all definitions and constructions mentioned below can be readily extended to covertext channels.

We borrow the complexity-theoretic notions of probabilistic polynomial-time algorithms and *negligible functions*, in terms of a security parameter n, from modern cryptography [4].

Definition 1 (Stegosystem). *Let $\mathcal{C}$ be a distribution on a set C of* covertexts. *A* stegosystem *is a triple of probabilistic polynomial-time algorithms* (SK, SE, SD) *with the following properties:*

- *The* key generation algorithm SK *takes as input the security parameter n and outputs a bit string sk, called the* [stego] key.
- *The* steganographic encoding algorithm SE *takes as inputs the security parameter n, the stego key sk and a* message $m \in \{0,1\}^l$ *to be embedded and outputs an element c of the covertext space C, which is called* stegotext. *The algorithm may access the covertext distribution $\mathcal{C}$.*
- *The* steganographic decoding algorithm SD *takes as inputs the security parameter n, the stego key sk, and an element c of the covertext space C and outputs either a message $m \in \{0,1\}^l$ or a special symbol $\perp$. An output value of $\perp$ indicates a decoding error, for example, when SD has determined that no message is embedded in c.*

For all sk output by $\mathsf{SK}(1^n)$ and for all $m \in \{0,1\}^l$, the probability that $\mathsf{SD}(1^n, sk, \mathsf{SE}(1^n, sk, m)) \neq m$ must be negligible in n.

Note that the syntax of a stegosystem as defined above is equivalent to that of a (symmetric-key) cryptosystem, except for the presence of the covertext distribution. The probability that the decoding algorithm outputs the correct embedded message is called the *reliability* of a stegosystem.

Defining Security: The security of a stegosystem is defined in terms of an experiment that measures the capability of the adversary to detect the presence of an embedded message. In a secure stegosystem, Eve cannot distinguish whether Alice is sending legitimate covertext or stegotext.

The attack considered here is a *chosen-message attack*, where the adversary may influence the embedded message but has otherwise no access to the encoding and decoding functions. It parallels the notion of a chosen-plaintext attack against a cryptosystem.

Consider an adversary defined by a pair of algorithms (SA_1, SA_2). The experiment consists of four stages.

1. A key sk is generated by running the key generation algorithm SK.
2. Algorithm SA_1 is run with input the security parameter n; it outputs a tuple (m^*, s), where $m^* \in \{0,1\}^l$ is a message and s is some additional information which the algorithm wants to preserve. SA_1 has access to the covertext distribution $\mathcal{C}$.
3. A bit b is chosen at random and a *challenge covertext* c^* is determined depending on it: If $b = 0$ then $c^* \leftarrow \mathsf{SE}(sk, m^*)$ (c^* becomes a steganographic encoding of m^*) otherwise $c^* \overset{R}{\leftarrow} \mathcal{C}$ (c^* is chosen randomly according to $\mathcal{C}$).

4. Algorithm SA_2 is run with inputs n, c^*, m^*, and s, and outputs a bit b'. The goal of SA_2 is to guess the value of b, i.e., to determine whether the message m^* has been embedded in c or whether c has simply been chosen according to $\mathcal{C}$.

The adversary succeeds to distinguish stegotext from covertext if $b' = b$ in the above experiment. Since it is trivial to achieve $\Pr[b' = b] = \frac{1}{2}$, what actually counts is the adversary's advantage above randomly guessing b. Formally, we define the *advantage* of adversary (SA_1, SA_2) to be

$$\Pr\Big[sk \leftarrow \mathsf{SK};\ (m^*, s) \leftarrow SA_1(1^n);\ b \overset{R}{\leftarrow} \{0, 1\};\ \textbf{if } b = 0 \textbf{ then } c^* \leftarrow \mathsf{SE}(1^n, sk, m^*)\ \textbf{else } c^* \overset{R}{\leftarrow} \mathcal{C} :\ SA_2(1^n, c^*, m^*, s) = b\Big] - \frac{1}{2}.$$

Depending on the detection capabilities ascribed to the adversary in terms of its computational power, and depending on its advantage in distinguishing stegotext from covertext, one obtains the following security notions for stegosystems:

Perfectly secure steganography: The adversary is an arbitrary, unbounded algorithm and has advantage equal to 0.

Statistically secure steganography: The adversary is an arbitrary, unbounded algorithm and has only negligible advantage (in n).

Computationally secure steganography: The adversary is an arbitrary probabilistic, polynomial-time computable algorithm and has only negligible advantage in n.

The first two notions are collectively known as *information-theoretically secure steganography*. The third notion, computational security, is discussed further on (cf., security).

This discussion has so far assumed that the sender and the receiver share the same secret key before communicating (cf., symmetric cryptosystem). Departing from that assumption, it is also possible to define *public-key steganography* with computational security. This notion is analogous to public-key cryptography and is the subject of Section D (cf., asymmetric cryptosystem).

For the scope of this survey, the adversary is limited to *passive attacks* (see cryptanalysis). In the Prisoners Problem, this means that the couriers may not change the messages communicated between Alice and Bob and that Eve may not send a message generated by herself to Bob and observe his reaction to it. However, the adversary may influence the messages to be embedded; for example, the warden Eve may determine the details of Alice and Bob's escape plan by choosing to confine them in particular cells.

This survey is about the formal approach to steganography and about stegosystems that offer provable security. An overview of steganography with heuristic security and of the history of steganography is given by Anderson and Petitcolas [1].

What distinguishes steganography from other forms of information hiding is the focus on merely detecting the *presence* of a hidden message. Watermarking and fingerprinting are two different problems of information hiding, where the existence of a hidden message is public knowledge. The focus in these areas is on hiding the message in perceptual data from an observer that is typically a human, and on embedding the message robustly so that it cannot be removed without significantly distorting the data itself. The difference between watermarking and fingerprinting is that watermarking supplies digital objects with an identification of origin and all objects are marked in the same way; fingerprinting, conversely, attempts to identify individual copies of an object by means of embedding a unique marker in every copy that is distributed to a user.

INFORMATION-THEORETICALLY SECURE STEGANOGRAPHY

DEFINITION 2 (Perfect Security). *Given a covertext distribution $\mathcal{C}$, a stegosystem $(\mathsf{SK}, \mathsf{SE}, \mathsf{SD})$ is called* perfectly secure *with respect to $\mathcal{C}$ if for any adversary (SA_1, SA_2) with unbounded computational power, the advantage in the experiment above is equal to 0.*

Perfect security for a stegosystem parallels Shannon's notion of *perfect security* for a cryptosystem [9] (cf., Shannon's model). The requirement that every adversary has no advantage implies that the distributions of the challenge c^* are equal in the two cases where it was generated from SE (when $b = 0$) and sampled from $\mathcal{C}$ (when $b = 1$). Hence, the adversary obtains *no* information about b because she only observes the challenge c^* and the distribution of c^* is statistically independent of b. Perfectly secure stegosystems were defined by Cachin [3].

Perfectly secure stegosystems exist only for a very limited class of covertext distributions. For example, if the covertext distribution is uniform, the one-time pad is a perfectly secure stegosystem as follows.

Assume the covertext $\mathcal{C}$ is uniformly distributed over the set of n-bit strings for some positive n and let Alice and Bob share an n-bit key sk with uniform distribution. The encoding function computes the bitwise XOR of the n-bit message m and sk, i.e., $\mathsf{SE}(1^n, sk, m) = m \oplus sk$; Bob can decode this by computing $\mathsf{SD}(1^n, sk, c) = c \oplus sk$. The resulting stegotext is uniformly distributed in the set of n-bit strings. The one-time pad stegosystem is used like this in visual cryptography [7].

For covertext distributions that do not admit perfectly secure stegosystems, one may still achieve the following security notion.

Definition 3 (Statistical Security). *Given a covertext distribution $\mathcal{C}$, a stegosystem* (SK, SE, SD) *is called* statistically secure *with respect to $\mathcal{C}$ if for all adversaries (SA_1, SA_2) with unbounded computational power, there exists a negligible function ϵ such that the advantage in the experiment above is at most $\epsilon(n)$.*

Statistical security for stegosystems may equivalently be defined by requiring that for any sk and any m, the statistical distance between the probability distribution generated by $\mathsf{SE}(1^n, sk, m)$ and the covertext distribution is negligible.

Definition 3 was first proposed by Katzenbeisser and Petitcolas [6]. A very similar notion was defined by Cachin [3], using relative entropy between the stegotext and covertext distributions for quantifying the difference between them.

Here is a simple example of a statistically secure stegosystem, adopted from [3]. It is representative for a class of practical stegosystems that embed information in a digital image by modifying the least significant bit of every pixel representation [1]. Suppose that the cover space C is the set of n-bit strings with (C_0, C_1) being a partition of C and with distribution $\mathcal{C}$ such $\left|\Pr[c \overset{R}{\leftarrow} \mathcal{C} : c \in C_0] - \Pr[c \overset{R}{\leftarrow} \mathcal{C} : c \in C_1]\right| = \delta(n)$ for some negligible δ. Then there is a stegosystem for a one-bit message m using a one-bit secret key sk. The encoding algorithm SE computes $s \leftarrow m \oplus sk$ and outputs $c \overset{R}{\leftarrow} C_s$. Decoding works without error because $m = 0$ if and only if $c \in C_{sk}$. It is easy to see that the encoding provides perfect secrecy for m and that the stegosystem is statistically secure. Note, however, that finding the partition for a given distribution is an NP-hard combinatorial optimization problem.

There exist also statistically secure *universal* stegosystems, where the covertext distribution is only available as a sampling oracle. Information-theoretically secure stegosystems suffer from the same drawback as cryptosystems with unconditional security in the sense that the secret key may only be used once. This is not the case for computational security considered next.

Computationally Secure Steganography

Definition 4. (Computational Security). *Given a covertext distribution $\mathcal{C}$, a stegosystem* (SK, SE, SD) *is called* computationally secure *with respect to $\mathcal{C}$ if for all probabilistic polynomial-time adversaries (SA_1, SA_2), there exists a negligible function ϵ such that the advantage in the experiment above is at most $\epsilon(n)$.*

The notion was formalized independently by Katzenbeisser and Petitcolas [6] and by Hopper, Langford, and von Ahn [5]. The latter work also presented the following construction of a computationally secure, universal stegosystem. It illustrates a popular encoding method that does not rely on knowledge of the covertext distribution, which is also used by some practical stegosystems.

The encoding method is based on an algorithm sample, which samples a covertext according to $\mathcal{C}$ such that a given bit string b of length $f = O(\log|C|)$ is embedded in it.

```
Algorithm sample
Input: security parameter $n$, a function $g : C \to \{0,1\}^f$, and a value $b \in \{0,1\}^f$
Output: a covertext $x$
  $j \leftarrow 0$
  repeat
      $x \overset{R}{\leftarrow} \mathcal{C}$
      $j \leftarrow j + 1$
  until $g(x) = b$ or $j = n$
  return $x$
```

Intuitively, algorithm sample returns a covertext chosen from distribution $\mathcal{C}$, but restricted to that subset of C which is mapped to the given b by g. sample may also fail and return a covertext c with $g(c) \neq b$, but this happens only with negligible probability in n.

Suppose $\{G_k\}$ is a pseudorandom function family indexed by k, with domain $\{0, 1\} \times C$ and range $\{0, 1\}^f$. (It can be thought of as a pair (G_0, G_1) of independent pseudorandom functions.) The secret key of the stegosystem consists of a randomly chosen k. The encoding algorithm $\mathsf{SE}(1^n, k, m)$ for an f-bit message m first "encrypts" m to $y \leftarrow G_k(0, c_0) \oplus m$ for a public constant $c_0 \in C$. Note that y is the ciphertext of a symmetric-key encryption of m and is computationally indistinguishable from a random f-bit string. This value y is then embedded by computing a stegotext $c \leftarrow \mathsf{sample}(n, G_k(1, \cdot), y)$. It can be shown that when

$\mathcal{C}$ is sufficiently random, as measured in terms of min-entropy, the output distribution of sample is statistically close to $\mathcal{C}$ [2, 5].

The decoding algorithm $\mathsf{SD}(1^n, k, c)$ outputs $m' \leftarrow G_k(1, c) \oplus G_k(0, c_0)$; it is easy to show that m' is equal to the message that was embedded using SE except with negligible probability.

This stegosystem is an extension of the example given above for statistical security. In fact, when G is a universal hash function and the encryption is realized using a one-time pad, this is a universal stegosystem with statistical security.

PUBLIC-KEY STEGANOGRAPHY: What if Alice and Bob did not have the time to agree on a secret key before being imprisoned? They cannot use any of the stegosystems presented so far because that would require them to share a common secret key. Fortunately, steganography is also possible without shared secrets, only with public keys, similar to public-key cryptography. The only requirement is that Bob's public key becomes known to Alice in a way that is not detectable by Eve.

Formally, a public-key stegosystem consists of a triple of algorithms for key generation, message encoding, and message decoding like a (secret-key) stegosystem, but the key generation algorithm now outputs a stego key pair (spk, ssk). The public key *spk* is made available to the adversary and is the only key needed by the encoding algorithm SE. The decoding algorithm SD needs the secret key ssk as an additional input.

DEFINITION 5 (Public-key Stegosystem). *Let* $\mathcal{C}$ *be a distribution on a set* C *of* covertexts. *A* public-key stegosystem *is a triple of probabilistic polynomial-time algorithms (SK, SE, SD) with the following properties:*

- *The* key generation algorithm *SK takes as input the security parameter n and outputs a pair of bit strings* (spk, ssk), *called the* [stego] public key *and the* [stego] secret key.
- *The* steganographic encoding algorithm *SE takes as inputs the security parameter n, the stego public key* spk *and a* message $m \in \{0, 1\}^l$ *and outputs a covertext* $c \in C$.
- *The* steganographic decoding algorithm *SD takes as inputs the security parameter n, the stego secret key ssk, and a covertext* $c \in C$, *and outputs either a message* $m \in \{0, 1\}^l$ *or a special symbol* $\perp$.

For all (spk, ssk) *output by the key generation algorithm and for all* $m \in \{0, 1\}^l$, *the probability that* $SD(1^n, ssk, SE(1^n, spk, m)) \neq m$ *must be negligible in n.*

Security is defined analogously to the experiment of Section 2 with the difference that the public key *spk* is additionally given to the adversary algorithms SA_1 and SA_2 and that the challenge covertext is computed using *spk* only. With these modifications, a public-key stegosystem (SK, SE, SD) is called *secure against chosen-plaintext attacks* if it is computationally secure according to Definition 4.

Secure public-key stegosystems can be constructed using the method of Section D, but with the pseudorandom function G_0 (which is used for "encryption") replaced by a public-key cryptosystem that has almost uniform ciphertexts. This property means that the output of the encryption algorithm is computationally indistinguishable from a uniform bit string of the same length.

The definition and several constructions of public-key stegosystems have been introduced by von Ahn and Hopper [11] and by Backes and Cachin [2]. The latter work also goes beyond the case of passive adversaries considered here and models adaptive chosen-covertext attacks, which are similar to adaptive chosen-ciphertext attacks against public-key cryptosystems. Achieving security against such attacks results in the strongest security notion known today for public-key cryptosystems and for public-key stegosystems.

As this brief survey of steganography shows, the evolution of the formal approach to stegosystems has gone through the same steps as the development of formal models for cryptosystems. The models and the formulation of corresponding stegosystems that offer provable security have greatly enhanced our understanding of this important area of information security.

Christian Cachin

References

[1] Anderson, R.J. and F.A. Petitcolas (1998). "On the limits of steganography." *IEEE Journal on Selected Areas in Communications*, 16.

[2] Backes, M. and C. Cachin (2005). "Public-key steganography with active attacks." *Proceedings 2nd Theory of Cryptography Conference (TCC 2005)*, Lecture Notes in Computer Science, vol. 3378, ed. J. Kilian. Springer, Berlin, 210–226.

[3] Cachin, C. (2004). "An information-theoretic model for steganography." Information and Computation, vol. 192, pp. 41–56. (Preliminary version appeared in Proc. 2nd Workshop on Information Hiding, Lecture Notes in Computer Science, vol. 1525, Springer, Berlin, 1998).

[4] Goldreich, O. (2001). *Foundations of Cryptography: Basic Tools*. Cambridge University Press, Cambridge.

[5] Hopper, N.J., J. Langford, and L. von Ahn (2002). "Provably secure steganography." *Advances in Cryptology—CRYPTO 2002*, Lecture Notes in Computer Science, vol. 2442, ed. M. Yung. Springer, Berlin.

[6] Katzenbeisser, S. and F.A.P. Petitcolas (2002). "Defining security in steganographic systems." *Security and Watermarking of Multimedia Contents IV, Proceedings of SPIE*, International Society for Optical Engineering, vol. 4675, eds. E.J. Delp and P.W. Won, 260–268.

[7] Naor, M. and A. Shamir (1995). "Visual cryptography." *Advances in Cryptology—EUROCRYPT'94*, Lecture Notes in Computer Science, vol. 950, ed. A. De Santis. Springer, Berlin, 1–12.

[8] Pfitzmann, B. (1996). "Information hiding terminology." *Information Hiding, First International Workshop*, Lecture Notes in Computer Science, vol. 1174, ed. R. Anderson. Springer, Berlin, 347–350.

[9] Shannon, C.E. (1949). "Communication theory of secrecy systems." *Bell System Technical Journal*, (28), 656–715.

[10] Simmons, G.J. (1984). "The prisoners' problem and the subliminal channel." *Advances in Cryptology—CRYPTO'83*, Lecture Notes in Computer Science, ed. D. Chaum. Plenum, New York, 51–67.

[11] von Ahn, L. and N.J. Hopper (2004). "Public-key steganography." *Advances in Cryptology–EUROCRYPT 2004*, Lecture Notes in Computer Science, vol. 3027, eds. C. Cachin and J. Camenisch. Springer, Berlin, 322–339.

DISCRETE LOGARITHM PROBLEM

Let G be a cyclic group of order n, and g be a generator for G. Given an element $y \in G$, the *discrete logarithm problem* is to find an integer x such that

$$g^x = y.$$

The discrete logarithm problem has been of particular interest since Diffie and Hellman (see Diffie–Hellman key agreement) invented a cryptographic system based on the difficulty of finding discrete logarithms (a similar system was created around the same time by Malcolm Williamson at the Government Communications Headquarters (GCHQ) in the UK, but not revealed until years later). Given two people Alice and Bob who wish to communicate over an insecure channel, each decides on a private key x_A and x_B. Alice sends g^{x_A} to Bob, and Bob sends g^{x_B} to Alice. Each of them can then raise the received message to their private key to compute

$$(g^{x_A})^{x_B} = (g^{x_B})^{x_A} = g^{x_A x_B}.$$

An eavesdropper Eve who only knows g^{x_A} and g^{x_B} must figure out $g^{x_A x_B}$. This is widely believed to be difficult. Clearly if Eve can solve the discrete logarithm problem, she can compute x_A and x_B and so break the system.

Other systems, such as the ElGamal digital signature scheme and the Digital Signature Standard, also depend on the difficulty of solving the discrete logarithm problem.

Pohlig and Hellman [9], and independently Silver, observed that if G has a subgroup of order l, then by raising g and y to the (n/l)th power we may solve for x modulo l. Thus, the difficulty of the discrete logarithm problem depends on the largest prime factor of n. For the rest of this article we will assume that n is prime.

THE DISCRETE LOGARITHM PROBLEM IN DIFFERENT GROUPS: Any finite group may be used for a Diffie–Hellman system, but some are more secure than others. The main groups used are:

- The multiplicative subgroup of a finite field GF(q), with q a prime or a power of 2.
- The points on an elliptic curve E over a finite field (see elliptic curves).
- The class group of a quadratic number field.

Finite fields GF(2^n) were popular into the 1980s, but attacks by Blake, Fuji-Hara, Mullin and Vanstone, and Coppersmith showed that the fields were easier to attack than similarly-sized prime fields. Index calculus attacks may also be applied to prime fields.

Hafner and McCurley [6] gave a subexponential attack for class groups of imaginary quadratic number fields, and Buchmann [2] extended this to real quadratic and, conjecturally, higher-degree number fields. Most elliptic curves, on the other hand, have no known subexponential attacks. See the entry on elliptic curve cryptography for more details.

GENERIC ALGORITHMS FOR DISCRETE LOGARITHMS: We will first consider *generic* algorithms, which do not use any special information about the group G, but only compose elements and check for equality. Nechaev [7] and Shoup [15] showed that generic algorithms must take $\Omega(\sqrt{n})$ time (see O-notation). Shor [14] showed that a quantum computer can solve a discrete logarithm problem in *any* group in polynomial time, but whether a sufficiently large quantum computer can be built is still an open problem.

Shanks' Baby Step–Giant Step Method

Shanks [13] gave the first algorithm better than a brute-force search. Let $m = \lceil\sqrt{n}\rceil$. We construct two tables, one starting at 1 and taking "giant steps" of length m:

$$1, g^m, g^{2m}, \ldots, g^{(m-1)m}$$

and one of "baby steps" of length one from y:

$$y, yg, yg^2, \ldots, yg^{m-1}.$$

Sort these lists and look for a match. If we find $g^{im} = yg^j$, then $y = g^{im-j}$, and so $x = im - j$. Any $x \in [0, n-1]$ may be written in this form for $i, j \leq m$, so we are certain to find such a match.

The time for this algorithm is $O(\sqrt{n})$ group operations, plus the time to find collisions in the two lists. This may be done either by sorting the lists or using hash tables.

Pollard's ρ Method

The drawback to Shanks's algorithm is that it requires $O(\sqrt{n})$ space as well as time. Pollard [10] gave two methods that use negligible space and still run in $O(\sqrt{n})$ time: the ρ method and the kangaroo method, which are discussed below. They are not deterministic, but depend on taking pseudorandom walks in G.

Divide the elements of G into three subsets, S_1, S_2 and S_3, say by the value of a hash of the elements modulo three. We define a walk by $h_0 = 1$ and

$$h_{i+1} = \begin{cases} h_i y, & \text{if } h_i \in S_1 \\ h_i^2, & \text{if } h_i \in S_2 \\ h_i g, & \text{if } h_i \in S_3. \end{cases}$$

At each step we know

$$h_i = g^{a_i} y^{b_i} = g^{a_i + x b_i}$$

for some a_i, b_i. (In particular, we have $(a_0, b_0) = (0, 0)$ initially, and $(a_{i+1}, b_{i+1}) = (a_i, b_i + 1)$, $(2a_i, 2b_i)$, or $(a_i + 1, b_i)$, depending on the hash value.) Eventually, this walk must repeat. If $h_i = h_j$, we have

$$x \equiv \frac{a_j - a_i}{b_i - b_j} \pmod{n}.$$

If $b_i - b_j$ is relatively prime to n (which is very likely if n is prime), this gives us x.

Figure 1 illustrates the ρ method walk.

Rather than store all of the steps to detect a collision, we may simultaneously compute h_i and h_{2i}, and continue around the cycle until they agree. Assuming that this map behaves as a random walk, we will need $O(\sqrt{n})$ steps to find a repeat.

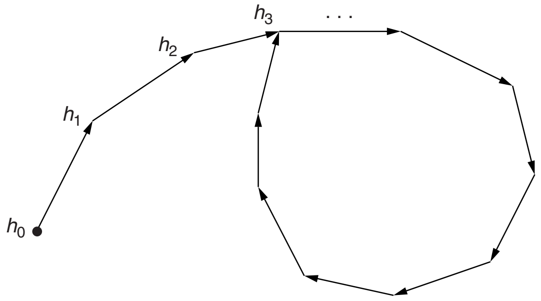

Fig. 1. ρ method walk, with a collision at h_3

Parallelized Collision Search

The ρ method has two main drawbacks. One is that it is difficult to parallelize. Having k processors do random walks only results in an $O(\sqrt{k})$ speedup, since the different walks are independent, and the probability of one of k cycles of length l having a collision is much less than one cycle of length kl (see [8] for details). Another is that after the collision occurs, many more steps around the cycle are needed before the collision is detected. Parallelized collision search [8] is a variant of the ρ method which fixes both problems.

We designate a small fraction of elements of G *distinguished points*, say if the last several bits of the element are all zero. Then a walk will begin at a random point, proceed as for the ρ method, and end when we hit a distinguished point. We save that point along with the starting point of the path, and then begin a walk at a new random point. When a distinguished point is hit for the second time, we have a collision and with high probability can determine x.

By picking the right fraction of elements of G to be distinguished points, we may ensure that not too much memory is needed to store the paths, and not much time is wasted after a collision occurs. Also, this algorithm may be trivially parallelized, with a linear speedup.

Figure 2 shows this method.

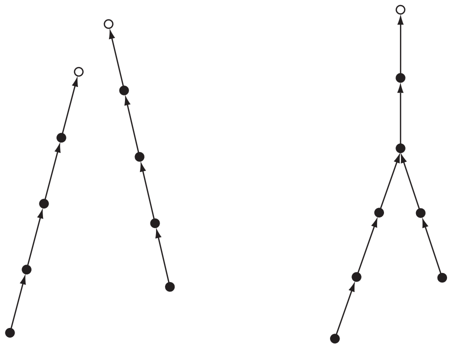

Fig. 2. Parallelized collision search paths, with three distinguished points and one collision

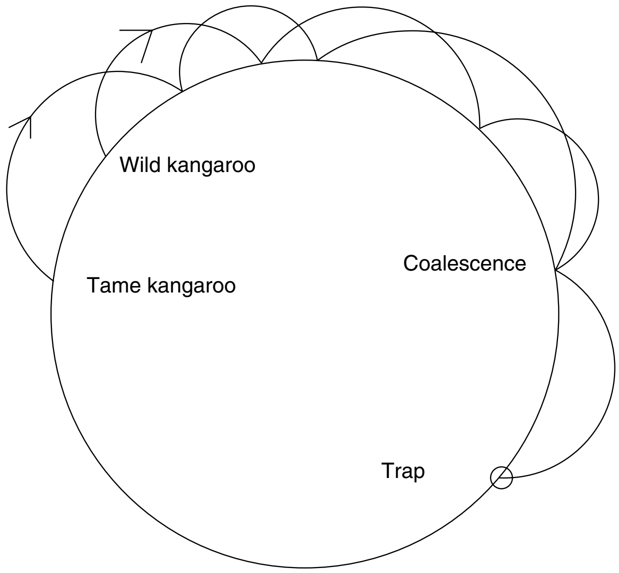

Fig. 3. Kangaroo method paths, with one distinguished point and collision

Pollard's Kangaroo Method

Another method due to Pollard also uses a random walk in G. In this algorithm the steps are limited: $h \longrightarrow hg^{s(h)}$, where the hop length $s(h)$ is a pseudorandom function of h with values between 1 and $\sqrt{n}$.

The idea is to start from two points, say g (the "tame" kangaroo, since we know its discrete logarithm at all times) and y (the "wild" kangaroo), and alternately take hops with length determined by $s(h)$. We will set "traps" when a kangaroo hits a distinguished point. If the wild kangaroo and tame kangaroo paths meet or "coalesce" at any point, they will take the same hops from then on, and any traps encountered by one after they coalesce will also be encountered by the other. When one reaches a trap that the other one hit, we have a collision and can determine x.

The main advantage of the kangaroo method is when x is known to be in a certain range, say $[0, L]$ for some $L \ll |G|$. In that case we may start the tame kangaroo from $g^{L/2}$, and the wild kangaroo from y. We expect to find a collision before we get far out of $[0, L]$, and so this will take $O(\sqrt{L})$ time. See Figure 3 for an illustration.

SUBEXPONENTIAL METHODS: The lower bound for generic algorithms means that to find a faster algorithm we must use information about the group. The main method for doing this is called *index calculus*, and is described in this section.

Index Calculus Methods

Let

$$L_x[t, \gamma] = e^{(\gamma+o(1))(\log x)^t(\log\log x)^{1-t}},$$

for $x \to \infty$ (see L-notation for further discussion). This function interpolates between slow and fast algorithms; $L_x[1, \gamma] \approx x^\gamma$ is exponential in $\log x$ (see exponential time), while $L_x[0, \gamma] \approx (\log n)^\gamma$ is polynomial (see polynomial time). All the algorithms of the previous section are $O(\sqrt{n}) = L_n[1, 1/2]$. With early index calculus methods we may reduce this to $L_n[1/2, c]$, and the number field sieve further improves this to $L_n[1/3, c]$ for appropriate constants c.

All index calculus algorithms for discrete logarithms have three main parts:

1. Gather equations relating the discrete logarithms of a factor base of "small" elements.
2. Solve a linear system to find the factor base discrete logarithms.
3. To find the discrete logarithm of an element y, reduce y to a product of elements in the factor base.

The first step is the same as in integer factoring. The second step is also done in factoring, but modulo 2 instead of modulo n. The third step is only done for discrete logarithms, typically by multiplying y by random powers of g, and attempting to express the result as a product of smaller numbers, possibly recursively breaking those numbers into smaller ones until everything is in the factor base.

The factor base is a set of elements such as small primes or low-degree polynomials, such that other elements have a reasonable chance of being "smooth": expressible as a product of these small elements (see the entry on smoothness). To optimize the algorithm we need to know the probability of this happening; see the section on number theory for more information.

Typically the first two steps require large computations, and finding individual logarithms is much quicker.

For additional technical details on these methods, please see the entry *index calculus*.

Discrete Logarithms in Prime Fields

Coppersmith, Odlyzko, and Schroppel gave an $L_p[1/2, c]$ algorithm for prime fields GF(p), which turned out to be special case of the Number Field Sieve (using imaginary quadratic fields). In their method there are two factor bases, one of small rational primes and another of small primes in the imaginary quadratic field.

The Number Field Sieve, which is the fastest known algorithm for factoring integers, may also be applied to finding discrete logarithms [5, 12]. The factor base used consists of small rational primes and representatives of small prime ideals in a number field. The asymptotic complexity is the same as for factoring. The sieving phase is the same, but solving the linear system modulo $p - 1$ instead of modulo 2 makes discrete logarithms harder than factoring problems of the same size.

Because the number field sieve works better for special numbers (such as primes $p = r^e + s$ for r and s small), it has been suggested that the Digital Signature Standard could be given a "trapdoor" by using a prime for which the Number Field Sieve runs faster than on a typical prime of that size. However, in [4], it is shown that such trapdoors may be detected, and that it is easy to specify primes which were clearly not chosen with a trapdoor.

Discrete Logarithms in Fields of Characteristic 2

Until the 1980s, fields GF(q) with $q = 2^n$ received the most attention, because of their applications to shift registers and ease of implementation in hardware. However, it turned out that attacks on these fields ran much faster than prime fields, and so few cryptosystems today depend on discrete logarithms in these fields.

Blake et al. [1] gave an attack which ran in time $L_q[1/2, c]$. Their factor base consists of polynomials in GF(2)[x] of low degree. This was improved by Coppersmith [3], who gave the first index calculus algorithm which runs in time $L_q[1/3, c]$. It was not realized until much later, but Coppersmith's method was a special case of the function field sieve (see the entry sieving in function fields).

Other Fields

Schirokauer [11] has looked at GF(q) for $q = p^m$ with $p > 2$ and $m > 1$. By combining features of the number field sieve and function field sieve, he gives an algorithm which is conjectured to run in $L_q[1/3, c]$ for some c for fields with $q \longrightarrow \infty$ and $m > (\log p)^2$ or $m < (\log p)^{1/2-\epsilon}$. In the "gap" between these constraints the algorithm is conjectured to run in time $L_q[2/5, c']$.

Recently Lenstra and Verheul invented a cryptosystem called XTR, which depends on the security of discrete logarithms in GF(p^6), for $p^6 \approx 1024$ bits. Weber [17] has computed discrete logarithms in fields GF(p^2) for small p.

ATTACKS ON ELLIPTIC CURVE DISCRETE LOGARITHMS: The elliptic curve discrete logarithm problem (ECDLP) was suggested as a basis for cryptosystems in 1985 by Neal Koblitz and Victor Miller. Because no subexponential attack was known for them, much shorter key sizes could be used.

Since then, several attacks on special elliptic curves have been developed, but no index calculus attack for general curves are known.

CHALLENGES AND ATTACKS: In 1989, Kevin McCurley gave a challenge problem. Let $q = (7^{149} - 1)/6$, and $p = 2 \times 739q + 1$. McCurley gave two numbers modulo p which equal 7^x and 7^y for some x and y, and issued a challenge to find 7^{xy}.

The form of p was intended to make it easy to show that p is prime and that 7 is a primitive root modulo p. Unfortunately, soon afterwards the number field sieve was discovered, which showed that the special form of this p made the system much less secure. The challenge was broken in 1998 by Weber and Denny [18] using the special number field sieve.

Joux and Lercier found discrete logarithms modulo a nonspecial 120-digit prime in 2001. For fields of characteristic 2, the record is GF(2^{607}), which was done in 2001 by Thomé [16].

In 1997 Certicom issued a series of ECDLP challenges. The problems ranged from easy (curves over 79-bit fields), to very difficult (359-bit fields). The largest challenge problem solved to date is a curve over GF(p) for a 109-bit prime p by a group at Notre Dame in 2002, using parallelized collision search.

Daniel M. Gordon

References

[1] Blake, I.F., R. Fuji-Hara, R.C. Mullin, and S.A. Vanstone (1984). "Computing logarithms in fields of characteristic two." *SIAM Journal of Algebraic and Discrete Methods*, 5, 276–285.

[2] Buchmann, Johannes (1990). "A subexponential algorithm for the determination of class groups and regulators of algebraic number fields." *Séminaire de Théorie des Nombres, Paris 1988–1989, Progr. Math.*, vol. 91, Birkhäuser, Boston, 27–41.

[3] Coppersmith, D. (1984). "Fast evaluation of discrete logarithms in fields of characteristic two." *IEEE Transactions on Information Theory*, 30, 587–594.

[4] Gordon, D.M. (1992). "Designing and detecting trapdoors in discrete log cryptosystems." *Advances in Cryptology—CRYPTO'92*, Lecture Notes

in Computer Science, vol. 740, ed. E.F. Brickell. Springer, Berlin, 66–75.

[5] Gordon, D.M. (1993). "Discrete logarithms in GF(p) using the number field sieve." *SIAM J. Discrete Math.*, 6, 124–138.

[6] Hafner, J. and K. McCurley (1989). "A rigorous subexponential algorithm for computation of class groups." *J. Amer. Math. Soc.*, 2 (4), 837–850.

[7] Nechaev, V.I. (1994). "On the complexity of a deterministic algorithm for a discrete logarithm." *Math. Zametki*, 55, 91–101.

[8] van Oorschot, P.C. and M.J. Wiener (1999). "Parallel collision search with cryptanalytic applications." *J. Cryptology*, 12, 1–28.

[9] Pohlig, S.C. and M.E. Hellman (1978). "An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance." *IEEE Trans. Info. Theory*, IT-24, 106–110.

[10] Pollard, J.M. (1978). "Monte Carlo methods for index computation (mod p)." *Mathematics of Computation*, 32, 918–924.

[11] Schirokauer, O. "The impact of the number field sieve on the discrete logarithm problem in finite fields." *Proceedings of the 2002 Algorithmic Number Theory workshop at MSRI*.

[12] Schirokauer, O. (1993). "Discrete logarithms and local units." *Philos. Trans. Roy. Soc. London Ser. A*, 345, 409–423.

[13] Shanks, D. (1971). "Class number, a theory of factorization, and genera." In *1969 Number Theory Institute (Proc. Sympos. Pure Math., Vol. XX, State Univ. New York, Stony Brook, NY, 1969)*, Amer. Math. Soc., Providence, RI, 415–440.

[14] Shor, P.W. (1997). "Polynomial-time algorithms for prime factorization and discrete logarrithms on a quantum computer." *SIAM J. Comput.*, 26, 1484–1509.

[15] Shoup, V. (1997). "Lower bounds for discrete logarithms and related problems." *Advances in Cryptolog—EUROCRYPT'97*, Lecture Notes in Computer Science, vol. 1233, ed. W. Furny. Springer, Berlin, 256–266.

[16] Thomé, E. (2001). "Computation of discrete logarithms in GF(2^{607})." *Advances in Cryptography—ASIACRYPT 2001*, Lecture Notes in Computer Science, vol. 2248, ed. C. Boyd. Springer, Berlin, 107–124.

[17] Weber, D. (1998). "Computing discrete logarithms with quadratic number rings." *Advances in Cryptology—EUROCRYPT'98*, Lecture Notes in Computer Science, vol. 1403, ed. K. Nyberg. Springer, Berlin, 171–183.

[18] Weber, D. and T.F. Denny (1986). "The solution of McCurley's discrete log challenge." *Advances in Cryptology—CRYPTO'98*, Lecture Notes in Computer Science, vol. 1462, ed. H. Krawczyk. Springer, Berlin, 458–471.