The mathematics of cryptology

Paul E. Gunnells

Department of Mathematics and Statistics
University of Massachusetts, Amherst
Amherst, MA 01003
www.math.umass.edu/∼gunnells

April 27, 2004

What is *Cryptology*?

- *Cryptography* is the process of writing using various methods ("ciphers") to keep messages secret.

- *Cryptanalysis* is the science of attacking ciphers, finding weaknesses, or even proving that a cipher is secure.

- *Cryptology* covers both; it's the complete science of secure communication.

Basic terminology/notation

- P is the *plaintext*. This is the original readable message (written in some standard language, like English, French, Cantonese, Hindi, Icelandic, . . .).

- C is the *ciphertext*. This is the output of some encryption scheme, and is not readable by humans.

- E is the *encryption function*. We write, for example,

$$E(P) = C$$

  to mean that applying the encryption process E to the plaintext P produces the ciphertext C.

- D is the *decryption function*, i.e.

$$D(C) = P.$$

  Note $D(E(P)) = P$ and $E(D(C)) = C$.

Basic terminology/notation (cont'd.)

- The *encryption key* is piece of data that allows the computation of E. Similarly we have the *decryption key*. These may or may not be the same. They also may not be secret, as we'll see later on.

- To *attack* a cipher is to attempt unauthorized reading of plaintext, or to attempt unauthorized transmission of ciphertext.

Shift (aka Cæsar) cipher

- Encode letters by numbers:

$$A \mapsto 0, B \mapsto 1, C \mapsto 2, \ldots, Z \mapsto 25.$$

- Choose a key t, which is a number between 0 and 25 (for Cæsar, t was always 3).

- For each letter P, E is defined by $E(P) = P + t$, i.e. add t to the code for each letter to get a new letter code. If you wind up with a number over 25, loop around to the beginning (like on a clock going past midnight). So, e.g. $25 + 3 = 2$.

Shift (aka Cæsar) cipher (cont'd.)

For example, if we take $t = 17$, then encrypting the plaintext

ALLOFGAULISDIVIDEDINTOTHREEPARTS

yields the ciphertext

RCCFWXRLCZJUZMZUVUZEKFKYIVVGRIJK

- Decryption is done by $D(C) = C - t$.

Remarks

- How did Cæsar get to rule the known Western world using this? It's horrendously insecure.

- Variations:
  - *Affine cipher*: Choose a number a and define $E(P) = aP + t$. Must be careful choosing a, e.g. $a = 0$ not very useful.
  - *Digraph affine cipher*: Choose numbers

    $$a_1, a_2, b_1, b_2, t_1, t_2,$$

    and then encrypt by transforming *pairs* of letters:

    $$E(P_1, P_2) = (a_1P_1 + b_1P_2 + t_1, a_2P_1 + b_2P_2 + t_2).$$

    Again the choices must be made carefully. But these schemes are still insecure, since natural languages have statistical biases (the *Wheel of Fortune* phenomenon).

Other uses of cryptography: Protocols

Today we use cryptography for a lot more than just sending secret messages.

- *Authentication*. Alice receives ciphertext from Bob. How can she be sure that the message originated from Bob? How can she be sure that the message wasn't corrupted? How can Bob be sure Alice received it? How can Alice make sure that Bob can't deny having sent it?

- *Key exchange*. Over an insecure channel, Alice and Bob exchange two pieces of data that allow them to compute a common encryption/decryption key. But any attacker who intercepts the transmissions can't recover the key.

Protocols (cont'd.)

- *Zero-knowledge proofs*. Alice can unequivocally convince Bob that she has a certain piece of information, without revealing the exact piece of information to Bob.

- *Secret sharing*. Alice, Bob, Carol, . . . , Yanni, and Zeke each have a piece of information that is part of a commonly held secret S.
  - If N or more of them meet and combine their knowledge, then S can be reconstructed.
  - But if less than N get together, S cannot be reconstructed.

All of these protocols are in common usage in computer networks today (ATMs, the Web, . . .). They are also crucial in sensitive communication between governments.

A modicum of mathematics

- *Integers*. Positive and negative counting numbers as well as 0, i.e. $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$.

- *Prime*. A positive integer that is divisible only by 1 and itself, e.g. $2, 3, 5, 7, 11, \ldots, 4136658067, \ldots$. The largest known prime today is $2^{20996011} - 1$, and has 6320430 digits. Integers that aren't prime are called *composite*.

- *Factoring*. Writing an integer as a product of smaller integers, e.g. $60 = 2^2 \cdot 3 \cdot 5$.

- *Primality test*. A test to decide whether or not an integer is prime. This is not the same as factoring.

- *Probabilistic primality test*. A test to decide whether or not an integer is prime to an (explicitly computable) high probability. Integers that pass these tests are called *pseudoprimes*, and in applications are just as useful as honest primes.

Public Key Cryptography

Each user has an encryption function and a decryption function.

- Alice makes her encryption function E_A publicly known, but keeps her decryption function D_A secret.
- Bob wants to send Alice a message P, so he computes $C = E_A(P)$ and sends it to her.
- Alice receives C and computes $P = D_A(C)$.

What makes this different from previous schemes, and why is it secure? The point is that the encryption/decryption functions are set up so that D_A is very difficult to compute only knowing E_A. Thus, even if an attacker knows E_A, he can't compute D_A and hence can't read Bob's message.

The RSA scheme

But how can it be that D_A isn't easily computable from E_A? How can there be such functions?

The RSA implementation of public key cryptography is based on the following empirically observed fact (here written as if it were carved in stone):

MULTIPLYING TWO INTEGERS IS EASY, BUT
FINDING A NONTRIVIAL FACTOR
OF AN INTEGER IS HARD.

In other words, integer multiplication is in practice a "one-way function." If a number is large, it's essentially impossible to factor it.

The RSA scheme (cont'd.)

- Alice secretly chooses two large primes p, q. Large means each has two hundred or so digits.

- She computes $N = p \cdot q$. Her encryption function E_A is built out of N, so she is essentially making N publicly known.[1]

- Her decryption function D_A, on the other hand, needs p and q to work. But knowing N isn't enough to figure out p and q!

[1] Actually she also makes another auxiliary integer e publicly known, but never mind. Knowledge of e doesn't help to figure out D_A.

Example of a protocol: Sender authentication

Each person has his own secret decryption function D, and everyone knows everyone else's encryption function D. Alice receives a message from Bob. How can she be sure it's really from him?

- At the very end of his message, Bob attaches an encrypted digital signature $S' = D_B(S)$.
  - S is a plaintext phrase ("Look at me; I'm as helpless as a kitten up a tree; Love, Bob"), but
  - S' looks like gibberish (it's been munged by the decryption function, which randomly garbles plaintext just as well as the encryption function).

  Hence the whole message P looks like normal plaintext with some junk at the end (S').

- Bob then computes $C = E_A(P)$ and sends it to Alice.

Example of a protocol: Sender authentication (cont'd.)

- Alice applies D_A to C and recovers P. She notes the gibberish S' at the end.

- Alice detaches S' and computes $E_B(S')$. (Recall that this function is publicly known).

- Since E_B undoes D_B, she gets as output S! Now she knows the message really came from Bob, since only he knows D_B, and to get S as $E_B(S')$ means that S' *must* have been computed using D_B.

- Finally she reads the actual message with signature. Blush, warm heartfelt glow, fade to black.

What does mathematics offer?

- Other sophisticated implementations of public key schemes
  - discrete logarithm schemes
  - elliptic curve cryptosystems
  - braid group cryptosystems

- Implementation improvements
  - effective primality tests
  - primality certificates
  - better versions of basic algorithms to speed up implementations

- Techniques to attack cryptosystems
  - advanced factoring methods such as the general number field sieve
  - other attacks based on statistical/probabilistic approaches

Recent accomplishments

A team led by Jens Franke of Bonn University (Germany) recently[2] factored the 174-digit RSA challenge number RSA-576

$$
\begin{array}{c}
1881 \\
9881292060\ 7963838697 \\
2394616504\ 3980716356\ 3379417382 \\
7007633564\ 2298885971\ 5234665485 \\
3190606065\ 0474304531\ 7388011303 \\
3967161996\ 9232120573\ 4031879550 \\
6569962213\ 0516875930 \\
7650257059
\end{array}
$$

into its two 87-digit prime factors. For this they won the not too shabby sum of $10,000. Such "challenges" are the only way we know that RSA is secure. In other words, factoring isn't *provably* hard, just *empirically* hard.

[2] December 3, 2003. In fact this was the day after the discovery of the largest known prime.

Appendix: How RSA works

- Alice chooses two primes p, q, and computes $N = pq$ and $\varphi(N) = (p-1)(q-1)$. She also chooses integers e, d such that $ed = 1 \mod \varphi(N)$.

- She publishes the pair (N, e). Her encryption function is $E_A(P) = P^e \mod N$ (we are assuming that plaintext is somehow encoded using integers mod N).

- Her decryption function is $D_A(C) = C^d \mod N$.

The scheme works because (by a result of Fermat from the 17th century) we have for any $P \mod N$

$$D_A(C) = D_A(P^e) = P^{ed} = P^{\varphi(N)} = P.$$

But to compute d from e and N, one needs to factor N.