Cryptography

Alessandro Tomasi
altomasi@fbk.eu

FONDAZIONE BRUNO KESSLER
CENTER FOR CYBERSECURITY
ALEPH
APPLIED CRYPTOGRAPHY

Diffie-Hellman

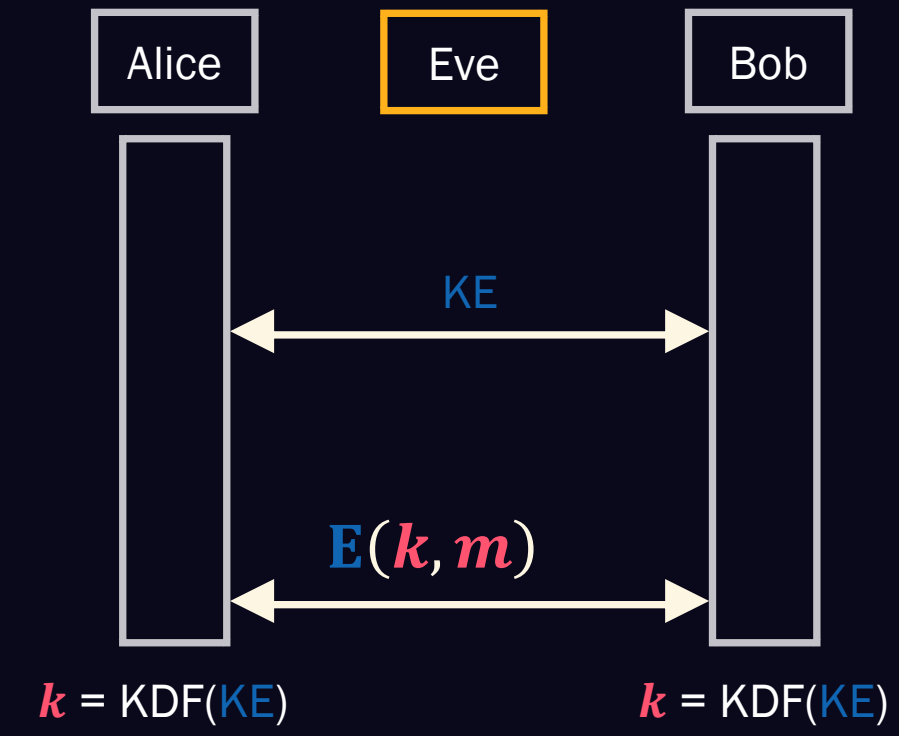
Key exchange [DH76]

FONDAZIONE BRUNO KESSLER
CENTER FOR CYBERSECURITY
ALEPH APPLIED CRYPTOGRAPHY

Key exchange

[DH76]

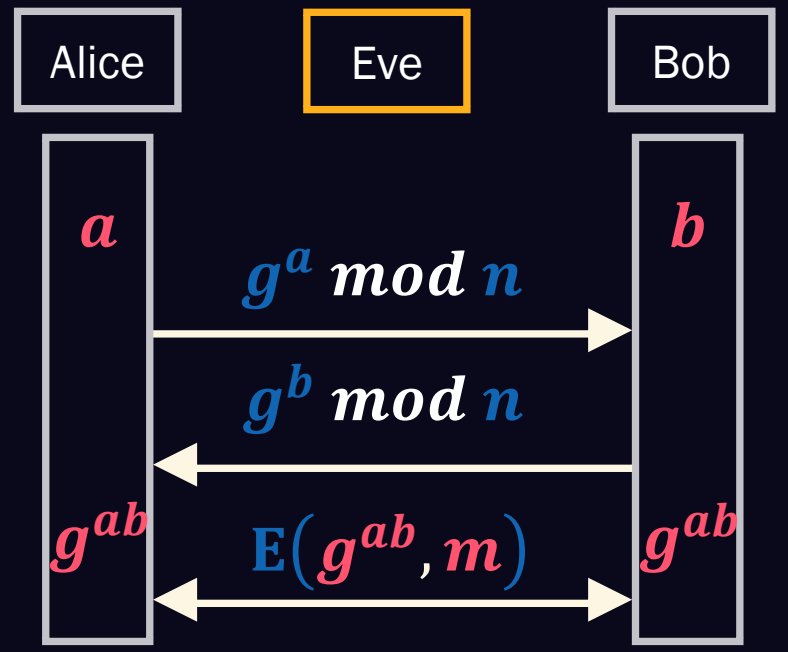
- Two parties, Alice and Bob, want to establish a common secret k.
- They can meet in person and exchange pre-shared keys (PSK)
- They can engage in an interactive protocol over an insecure, public channel, and derive a key from the transcript KE.



Diffie-Hellman Key Exchange

[DH76]

1. Public parameter agreement: n prime, g generator of a subgroup of order $q|(n-1)$
2. Independently, Alice draws random $a \in \mathbb{Z}_n^*$; Bob draws random $b \in \mathbb{Z}_n^*$. a and b are kept secret.
3. Alice and Bob exchange $g^a \ mod \ n$, $g^b \ mod \ n$ over an insecure, public channel.
4. Alice and Bob can each compute $g^{ab} \equiv (g^a)^b \equiv (g^b)^a \ mod \ n$



Diffie-Hellman Key Exchange

[DH76]

1. Public parameter agreement: n prime, g generator of a subgroup of order $q|(n-1)$
2. Independently, Alice draws random $a \in \mathbb{Z}_n^*$; Bob draws random $b \in \mathbb{Z}_n^*$. a and b are kept secret.
3. Alice and Bob exchange $g^a \ mod \ n$, $g^b \ mod \ n$ over an insecure, public channel.
4. Alice and Bob can each compute $g^{ab} \equiv (g^a)^b \equiv \left(g^b\right)^a \ mod \ n$

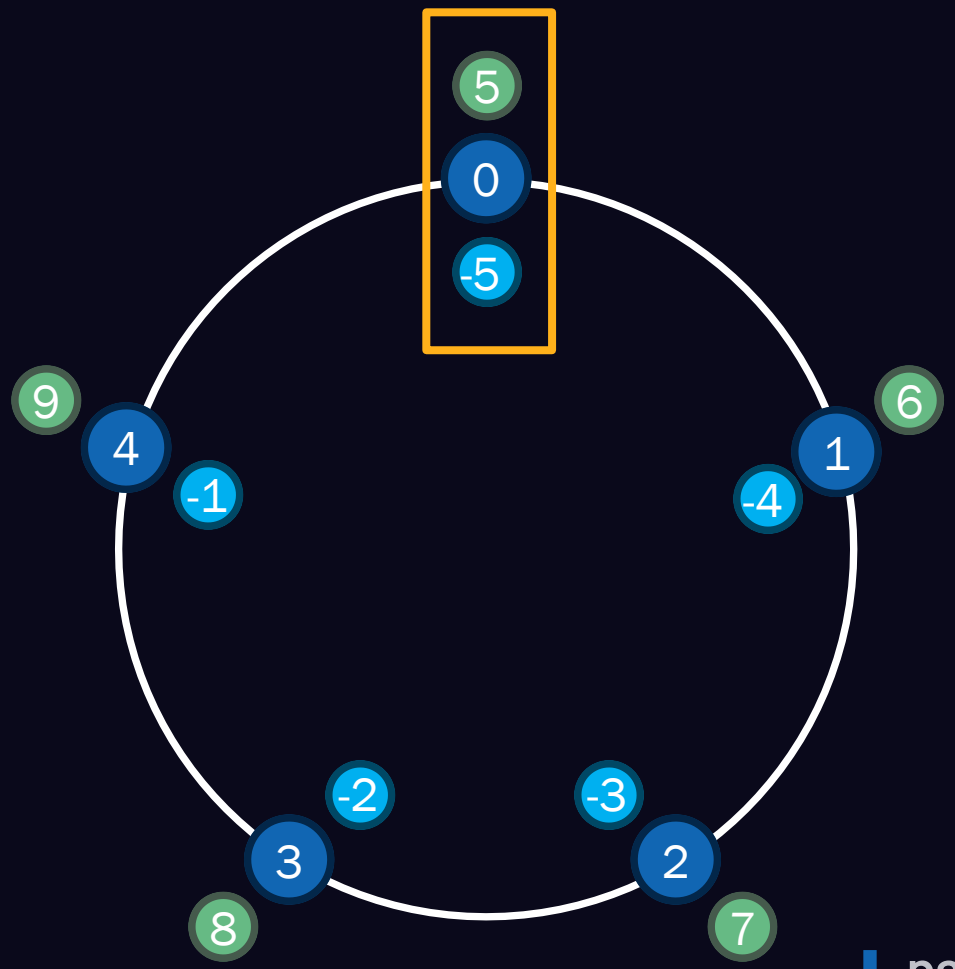
1. Public standards, e.g., [SP 800-56a], [RFC7919]
2. DHE (Ephemeral): new key for every exchange (forward secrecy).
3. Protocols define who sends which key, and when.
4. Protocols define how the encryption key may be derived from the established secret, e.g., TLS 1.2 [RFC5264], 1.3 [RFC8446]

Why does it work, and when does it **not**?

- Modular arithmetic and congruence
- Group, subgroup, generator, order
- Discrete Logarithm and Diffie-Hellman problem
- Useful for ECDH and RSA and LCG, too
- [MvOV01, BS23, H23]

Modular arithmetic

- $a \pmod{n} = r$
  - *remainder after division of* a *by* n
- a is congruent to b (modulo n) if and only if n divides $a - b$
  - $a \equiv b \pmod{n} \quad iff \quad n|a-b$
  - $a - b = kn$ *for some* k
- The set of all integers $a + kn \; \forall k$ is the **(congruence | residue) class** modulo n of a
  - *all integers with the same remainder as* a *after division by* n
- Integers $mod\ n$ form an additive group $\mathbb{Z}_n$



Group

- A **set** $\mathbb{G}$ with an **operation** $\circ$ such that:
  - *closed under* $\circ$: $x_1 \circ x_2 \in \mathbb{G}$ $\forall x_1, x_2 \in \mathbb{G}$
  - $\exists$ *identity* $i \in \mathbb{G}$: $i \circ x = x$ $\forall x \in \mathbb{G}$
  - $\exists$ *inverse* $x^{-1} \in \mathbb{G}$: $x^{-1} \circ x = i$ $\forall x \in \mathbb{G}$
  - *Associative:* $(x_1 \circ x_2) \circ x_3 = x_1 \circ (x_2 \circ x_3)$

Group

- A set $\mathbb{G}$ with an operation $\circ$ such that:
  - *closed under* $\circ$: $x_1 \circ x_2 \in \mathbb{G}$ $x_2 \circ x_1 \in \mathbb{G}$ $\forall x_1, x_2 \in \mathbb{G}$
  - $\exists$ *identity* $i \in \mathbb{G}$: $i \circ x = x$ $= x \circ i$ $\forall x \in \mathbb{G}$
  - $\exists$ *inverse* $x^{-1} \in \mathbb{G}$: $x^{-1} \circ x = i$ $= x \circ x^{-1}$ $\forall x \in \mathbb{G}$
  - *Associative:* $(x_1 \circ x_2) \circ x_3 = x_1 \circ (x_2 \circ x_3)$
- Abelian group:
  - *commutative:* $x_1 \circ x_2 = x_2 \circ x_1$ $\forall x_1, x_2 \in \mathbb{G}$
- Often implied unless otherwise stated as non-Abelian

Group

- A set $\mathbb{G}$ with an operation $\circ$ such that:
  - *closed under* $\circ$: $x_1 \circ x_2 \in \mathbb{G}$ $\forall x_1, x_2 \in \mathbb{G}$
  - $\exists$ *identity* $i \in \mathbb{G}$: $i \circ x = x$ $\forall x \in \mathbb{G}$
  - $\exists$ *inverse* $x^{-1} \in \mathbb{G}$: $x^{-1} \circ x = i$ $\forall x \in \mathbb{G}$
  - *Associative:* $(x_1 \circ x_2) \circ x_3 = x_1 \circ (x_2 \circ x_3)$
- Cyclic group: **every** other element of the group may be obtained by repeatedly applying the group operation to a generator element
  - $\exists$ *at least one generator:* $\mathbb{G} = (\langle g \rangle, \circ)$

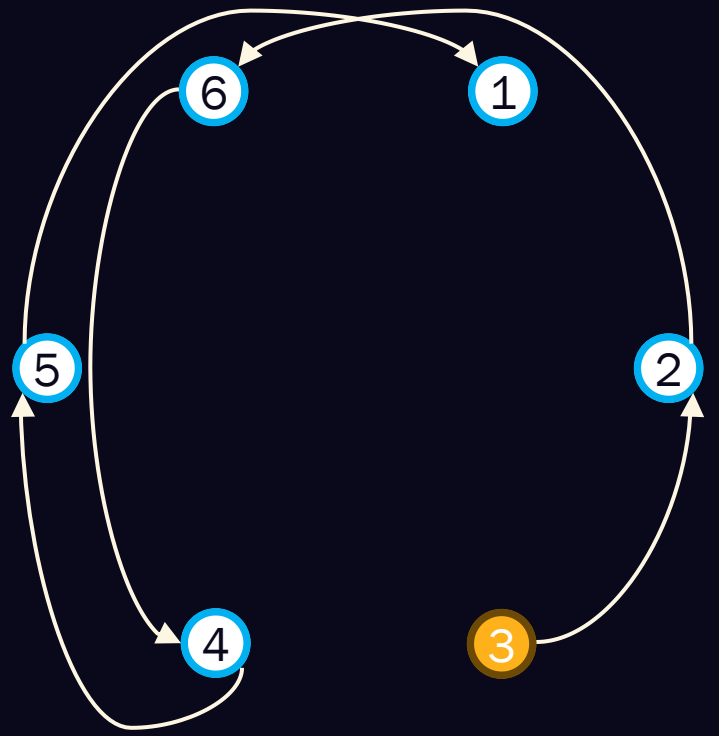
Examples

- $(\mathbb{Z}_n, +)$ is a cyclic group generated by 1
- The positive integers mod n, $\mathbb{Z}_n^*$: $\{x \in [1, n-1]\}$ is a multiplicative cyclic group $(\mathbb{Z}_n^*, \cdot)$ if $n \in \{1,2,4, p^k, 2p^k\}$, p an odd prime
  - *Closed:* $x_1 \cdot x_2 \in \mathbb{Z}_n^*$
  - *Identity:* $i \cdot x = x$
  - *Inverse:* $\exists\, x^{-1}\colon x^{-1} \cdot x = i$
  - *Generator:* $g^k \equiv x \pmod n$ *(at least one)*
- The solutions to Elliptic Curve equations E over finite fields $\mathbb{F}_p$ form an additive cyclic group
  - $y^2 = x^3 + ax + b$, *with* $a, b \in \mathbb{F}_p$, $4a^3 + 27b \neq 0$

Generator

- Example: 3 and 5 are both generators of $\mathbb{Z}_7^*$
- Order: hard to guess; element: hard to invert

3^1	3
3^2	$9 \equiv 2 \ (mod\ 7)$
3^3	$27 \equiv 6 \ (mod\ 7)$
3^4	$81 \equiv 4 \ (mod\ 7)$
3^5	$243 \equiv 5 \ (mod\ 7)$
3^6	$729 \equiv 1 \ (mod\ 7)$



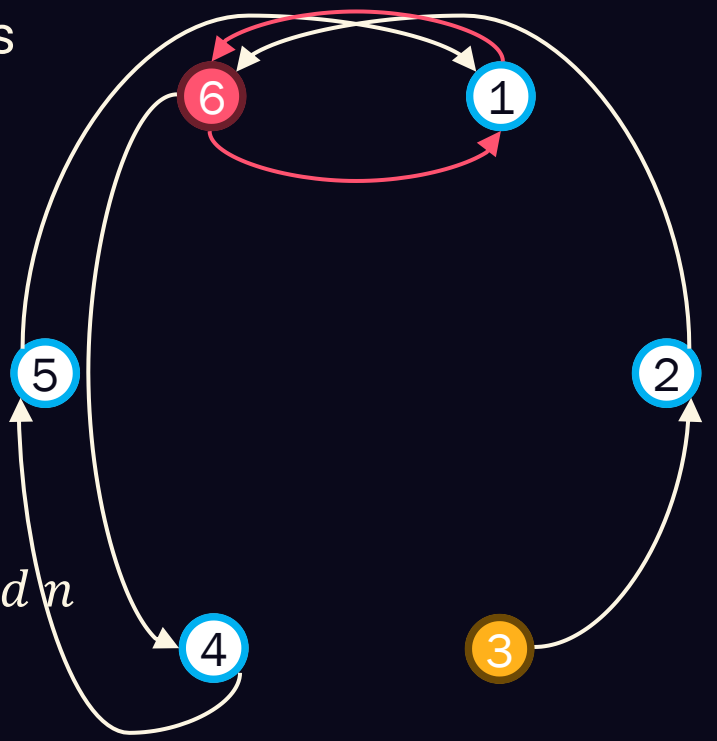
https://en.wikipedia.org/wiki/Primitive_root_modulo_n

Subgroups

- Smaller groups may be generated by other elements
- Example: 6 is a generator of $\{6,1\} \bmod 7$

6^0	1
6^1	$6 \equiv -1 \ (mod\ 7)$
6^2	$36 \equiv 1 \ (mod\ 7)$
6^3	$216 \equiv -1 \ (mod\ 7)$

- Beware: $n-1$ is always a generator of $\{n-1,1\}\ mod\ n$



https://en.wikipedia.org/wiki/Primitive_root_modulo_n

Order, generators, subgroups

- The order of a group is the smallest positive integer o satisfying $g^o \equiv 1 \, mod \, p$. Equivalently, it is the number of distinct elements of the subgroup $\{g, g^2, g^3, \ldots mod \, p\}$
- There are $\varphi(n)$ **elements** in $\mathbb{Z}_n^*$ (φ Euler's totient function)
- Special case: if n is the product of two primes, $n = pq$,

$$\varphi(n) = (p-1)(q-1)$$

- There are $\varphi(\varphi(n))$ **generators**, if any.

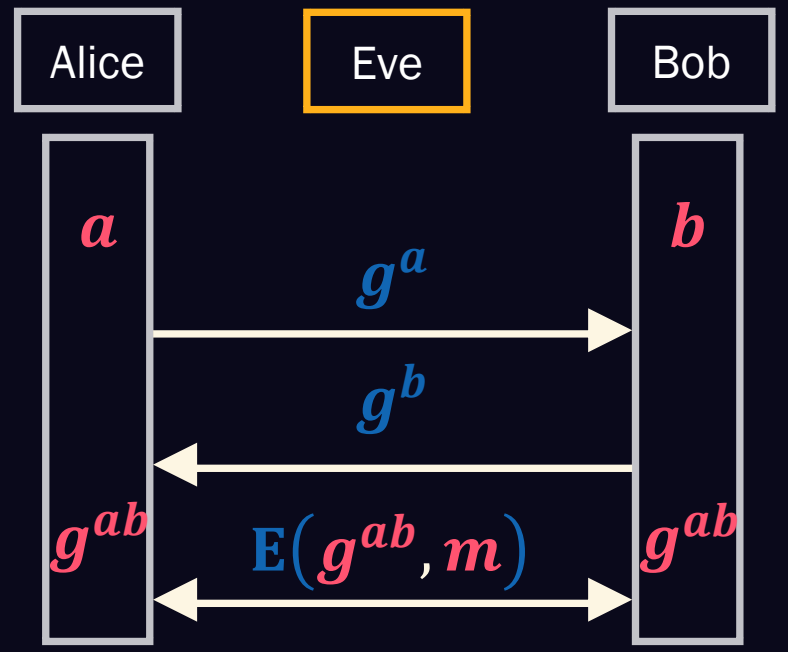
Problems assumed to be hard to solve

- Discrete Logarithm Problem (DLP): Given g of $\mathbb{Z}_n^*$ and y, find x such that $g^x \equiv y \mod n$
  - $y = \log_g x \pmod n$ *is the discrete logarithm of* x *to the base* g *modulo* n.
  - *"Inverse" of modular exponentiation.*
- Generalized DLP: given a finite cyclic group $\mathbb{G}$ of order n, a generator g of $\mathbb{G}$, and an element $y \in \mathbb{G}$, find the integer $x \in [0, n-1]$ such that $g^x = y$.
- No **efficient** classical algorithm is known for computing discrete logarithms. Best known: number field sieve (NFS). Some algorithms are not guaranteed to converge or to find all solutions [GJ21].
- Computational Diffie-Hellman (CDH): Given g, g^x, g^y, find g^{xy}. Hard to verify.
- Decisional Diffie-Hellman (DDH): Given g, g^x, g^y, distinguish g^{xy} from a random group element.
- DL easy ⇒ CDH false; CDH hard ⇐ DDH hard

Diffie-Hellman

[DH76]

1. Public parameters:
   1. $\mathbb{G}$ *a cyclic group with efficient exponentiation and hard DLog*
   2. g *a generator of* $\mathbb{G}$
2. Independently
   1. *Alice draws random secret* $a \in \mathbb{Z}$
   2. *Bob draws random secret* $b \in \mathbb{Z}$
3. Alice and Bob exchange g^a, g^b over an insecure, public channel.
4. Alice and Bob can each compute g^{ab}



References

- [BS23] A Graduate Course in Applied Cryptography. Dan Boneh and Victor Shoup, version 0.6, Jan. 2023. https://toc.cryptobook.us/
- [DH76] “New directions in cryptography”. W Diffie, M Hellman. IEEE Trans. Inf. Th. 22 (6): 644–654, 1976. doi: 10.1109/tit.1976.1055638
- [GJ21] “Computing Discrete Logarithms”. Robert Granger and Antoine Joux. 2021. eprint.iacr.org/2021/1140
- [H23] “Applied Cryptography CSE 207B” Nadia Heninger, Fall 2023 https://cseweb.ucsd.edu/classes/fa23/cse207B-a/
- [MvOV01] “Handbook of Applied Cryptography”, cacr.uwaterloo.ca/hac/

References

- [RFC5264] "The Transport Layer Security (TLS) Protocol Version 1.2". https://datatracker.ietf.org/doc/html/rfc5246
- [RFC7919] "Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for Transport Layer Security (TLS)". datatracker.ietf.org/doc/html/rfc7919
- [RFC8446] "The Transport Layer Security (TLS) Protocol Version 1.3". https://datatracker.ietf.org/doc/html/rfc8446
- [SP 800-56a] "Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography" revision 3, 2018. doi: 10.6028/NIST.SP.800-56Ar3. url: csrc.nist.gov/publications/detail/sp/800-56a/rev-3/final

Attacks on DH

Computing Discrete Logarithms [GJ21]
RSA, DH, and DSA in the Wild [H22]

FONDAZIONE BRUNO KESSLER
CENTER FOR CYBERSECURITY
ALEPH APPLIED CRYPTOGRAPHY

[logjam] (2015): downgrade to export grade cipher suites 512-bit DH

- TLS MitM downgrade connections to “export-grade” 512-bit Diffie-Hellman.
- “82% of vulnerable servers use a single 512-bit group - 7% of Alexa Top Million HTTPS sites. In response, major browsers are being changed to reject short groups.”
- “A small number of fixed or standardized groups are used by millions of servers; performing precomputation for a single 1024-bit group would allow passive eavesdropping on 18% of popular HTTPS sites, and a second group would allow decryption of traffic to 66% of IPsec VPNs and 26% of SSH servers.”

Bits	Precomputation for $\mathbb{G}$ (core-years)	~3k cores (2015)	Discrete Log	36 cores (2015)
512	10	~1 week	10 core-min’s	~70 sec’s
768	35,000		2 core-days	
1024	45,000,000		30 core-days	

GDLP in group $\mathbb{G}$ of order q, generator g

[MvOV01]#3.6.2
[V+17]

- **Exhaustive search.** Compute every g^j. Running time $\mathcal{O}(q)$ group multiplications.
- **Small order groups: Baby-step giant-step** [S71]. Running time $\mathcal{O}(\sqrt{q})$ group multiplications, storage $\mathcal{O}(\sqrt{q})$ group elements.
- **Small prime order groups: Pollard's rho** [P75]. Expected running time $\mathcal{O}(\sqrt{q})$ group operations, storage negligible, but starts with randomized initial values and may terminate with failure.
- **Composite-order groups.** If the group order q is a composite with prime factorization $q = \prod_i q_i^{e_i}$, use the Pohlig-Hellman algorithm [PH78] to compute a discrete log in time $\mathcal{O}(\sum_i e_i \sqrt{q_i})$.
- **Short exponents.** If the secret exponent a is relatively small or lies within a known range of values of a relatively small size, m, then the Pollard lambda "kangaroo" algorithm [P00] can be used to find a in time $\mathcal{O}(\sqrt{m})$
- **Small prime moduli in certain groups.** The best-known algorithm for discrete logarithms is the Number Field Sieve (NFS), running in $\approx \mathcal{O}\left(e^{\sqrt[3]{n}}\right)$, with n the bit length of p.

```
from sage.all import *
private_key = public_key.log(g)
```

Baby-step giant-step

[S71]?
[MvOV01]#3.6.2

- Find y such that $g^y = h$ in a cyclic group of order n with generator g. A more efficient way of searching through a space of size n than computing every element.
- Write $y = ax + b$, with $b < x$, setting $x = \lceil \sqrt{n} \rceil$; find a, b by trial.
  - $g^{ax+b} = h$
  - $g^b = h(g^{-x})^a$
- Compute a table (j, g^j), $j \in (0, x-1)$ (the baby steps), sort by g^j
- Compute $h(g^{-x})^i$, $i \in (0, x-1)$ (the giant steps) until you find a collision
  - $h(g^{-x})^i = g^j$
- Then, $ix + j = \log_g h$.

Example (Baby-step giant-step)

- Solve $3^y \equiv 2 \ (mod\ 29)$
- Order: $n = \varphi(29) = 28, x = \lceil \sqrt{n} \rceil = 6$
- Baby Steps g^j

j	0	1	2	3	4	5
3^j	1	3	9	27	23	11

- $g^{-x} = 3^{-6} \equiv 3^{28-6} \equiv 22 \ (mod\ n)$

Fermat's little theorem: if prime p does not divide a, then $a^{p-1} \equiv 1 \ (mod\ p)$

```
pow(3,-6,29)
```

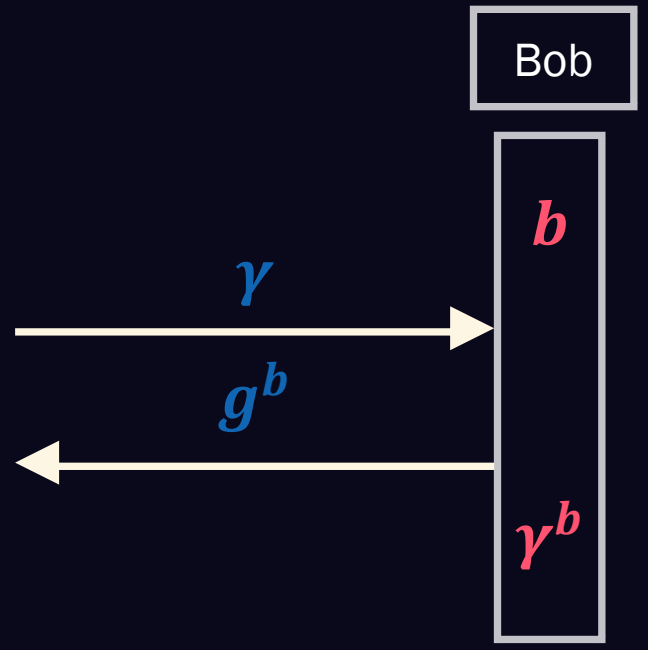
- Giant Steps $h(g^{-x})^i$

i	0	1	2
$2 \cdot 22^i$	2	15	11

$y = ix + j = 17$

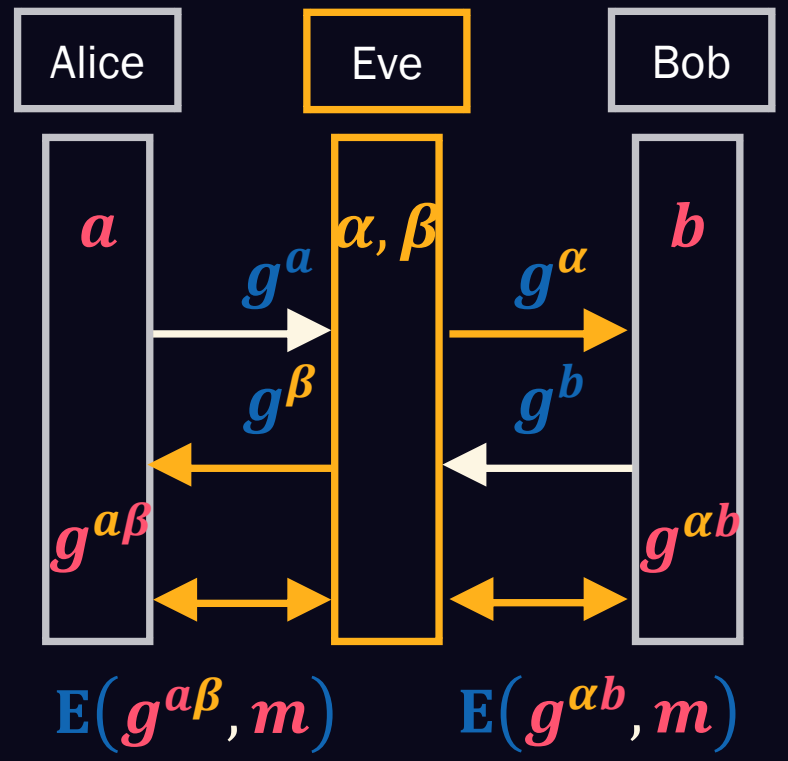
Bob’s view

- Bob’s view of the shared secret is computed with input



Man in the Middle

- If the eavesdropper can not only read messages on the public channel, but also intercept them, Eve may establish separate keys g^{Ab} with Bob and g^{aB} with Alice
- Eve can decrypt, read, and re-encrypt; Alice and Bob may never notice as long as Eve actively relays.
- Alice and Bob need authentication before key exchange.



Small subgroup attacks - confinement

- **Confinement**. [vOW96] A malicious Alice provides Bob with a key-exchange value g^a that lies in a subgroup of small order. This forces Bob's view of the shared secret, $(g^a)^b$, to lie in the subgroup generated by the attacker.
- **Zero**: in prime-field DH $\mathbf{0} \notin \boldsymbol{Z_p^*}$: Alice sends $g^a = 0$. Bob will always compute $(g^a)^b = 0$.

Small subgroup attacks - confinement

- **Confinement**. [vOW96] A malicious Alice provides Bob with a key-exchange value g^a that lies in a subgroup of small order. This forces Bob's view of the shared secret, $(g^a)^b$, to lie in the subgroup generated by the attacker.
- **Zero**: in prime-field DH $\mathbf{0} \notin \boldsymbol{Z_p^*}$: Alice sends $g^a = 0$. Bob will always compute $(g^a)^b = 0$.
- **Subgroup of Order 1**: Alice sends $g^a = 1$. Bob will always compute $(g^a)^b = 1$.

Small subgroup attacks - confinement

- **Confinement**. [vOW96] A malicious Alice provides Bob with a key-exchange value g^a that lies in a subgroup of small order. This forces Bob's view of the shared secret, $(g^a)^b$, to lie in the subgroup generated by the attacker.
- **Zero**: in prime-field DH $\mathbf{0} \notin \boldsymbol{Z_p^*}$: Alice sends $g^a = 0$. Bob will always compute $(g^a)^b = 0$.
- **Subgroup of Order 1**: Alice sends $g^a = 1$. Bob will always compute $(g^a)^b = 1$.
- **Subgroup of Order 2**: Alice sends g_2, the generator of the subgroup of size 2, e.g., $g^a = p - 1 \equiv -1 \ (mod\ p)$.
  - *Bob will always compute one of two elements, e.g.,*

$$(p-1)^b \ (mod\ p) = \begin{cases} 1 \\ -1 \end{cases}$$

Small subgroup attacks - confinement

- **Confinement**. [vOW96] A malicious Alice provides Bob with a key-exchange value g^a that lies in a subgroup of small order. This forces Bob's view of the shared secret, $(g^a)^b$, to lie in the subgroup generated by the attacker.
- **Zero**: in prime-field DH $\mathbf{0} \notin \boldsymbol{Z_p^*}$: Alice sends $g^a = 0$. Bob will always compute $(g^a)^b = 0$.
- **Subgroup of Order 1**: Alice sends $g^a = 1$. Bob will always compute $(g^a)^b = 1$.
- **Subgroup of Order 2**: Alice sends g_2, the generator of the subgroup of size 2. Bob will always compute one of two elements
- [V+17]

Server \ Accepts	0	1	-1
HTTPS	0.06 %	3 %	5 %
SSH	3 %	25 %	34 %

Subgroup attacks

- **Confinement**. [vOW96] A malicious Alice provides Bob with a key-exchange value g^a that lies in a subgroup of small order. This forces Bob's view of the shared secret, $(g^a)^b$, to lie in the subgroup generated by the attacker.
- **Key recovery**. If Alice learns a value derived from Bob's view of the shared secret – e.g., a ciphertext or MAC – then Alice can learn 1+ bits of information about Bob's secret exponent. [LL97]
  - *Bob reuses the secret exponent for multiple connections*
  - *Alice finds many small subgroups of order q_i modulo Bob's choice of DH prime p*
  - *Alice sends generators g_{q_i} of each subgroup order in sequence, and receives a value derived from Bob's view of the secret shared key in return.*
  - *Alice uses this information to recover Bob's secret exponent $b \bmod q_i$ for each q_i.*

Safe primes and groups

- To maximize the size of the subgroup used for Diffie-Hellman, one can choose a p such that $p = 2q + 1$ for some prime q. Such a $\boldsymbol{p}$ is called a **safe prime**, and such a q is called a Sophie Germain prime.
- If p is a safe prime, a common recommendation is to use a generator g of the subgroup of order q modulo p, not the full group of order $p - 1$.
- Some implementations choose small q e.g., 256 bits, then set $p \leftarrow qh + 1$ and choose g so it generates group of order q
- Implementations must check that received key-exchange values y are in the correct subgroup of order q by checking that $y^q \equiv 1 \; mod \; p$.
- For sufficiently large safe primes, the best attack will be solving the discrete log, e.g., using the NFS.

Example: ffdhe2048

[RFC7919#appendix-A.1]

- The 2048-bit group has registry value 256
  - *The modulus is* $p = 2^{2048} - 2^{1984} + (\lfloor 2^{1918} e \rfloor + 560316)2^{64} - 1$
  - *The generator is:* $g = 2$
  - *The group size is:* $q = (p - 1)/2$
- The estimated symmetric-equivalent strength of this group is 103 bits.
- Peers using ffdhe2048 that want to optimize their key exchange with a short exponent should choose a secret key of at least 225 bits.
- Traditional FFDH has each peer choose their secret exponent from the range $(2, p - 2)$. Using exponentiation by squaring, this means each peer must do roughly $2 \log_2(p)$ multiplications, twice (once for the generator and once for the peer's public key).

References

- [GJ21] “Computing Discrete Logarithms”. Robert Granger and Antoine Joux. 2021. eprint.iacr.org/2021/1140
- [H22] “RSA, DH and DSA in the Wild”. Nadia Heninger, 2022. eprint.iacr.org/2022/048
- [logjam] Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice. D Adrian, K Bhargavan, Z Durumeric, P Gaudry, M Green, J. A Halderman, N Heninger, D Springall, E Thomé, L Valenta, B VanderSloot, E Wustrow, S Zanella-Béguelin, P Zimmermann. 22nd ACM Conference on Computer and Communications Security (CCS ’15), Denver, CO, October 2015. doi: 10.1145/2810103.2813707, https://weakdh.org/, https://youtu.be/mS8gm-_rJgM.

References

- [LL97] "A key recovery attack on discrete log-based schemes using a prime order subgroup." C. H. Lim and P. J. Lee. In 17th International Cryptology Conference, 1997. doi: 10.1007/BFb0052240
- [MvOV01] "Handbook of Applied Cryptography", cacr.uwaterloo.ca/hac/
- [P75] "A Monte Carlo method for factorization." J. M. Pollard. BIT Numerical Mathematics, 15(3):331–334, 1975. doi: 10.1007/bf01933667
- [P00] "Kangaroos, Monopoly and discrete logarithms." M. J. Pollard. Journal of Cryptology, 2000. doi: 10.1007/s001450010010
- [PH78] "An improved algorithm for computing logarithms over GF(p) and its cryptographic significance." S. C. Pohlig and M. E. Hellman. IEEE Transactions on Information Theory, 24(1), 1978. doi: 10.1109/TIT.1978.1055817

References

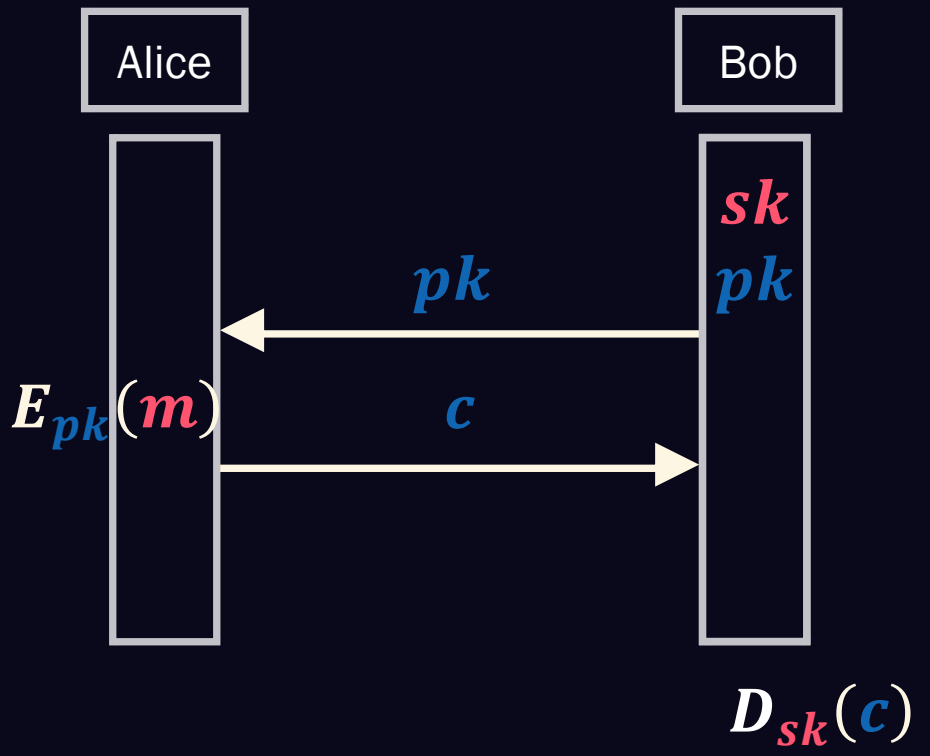
- [RFC7919] "Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for Transport Layer Security (TLS)". datatracker.ietf.org/doc/html/rfc7919
- [S71] "Class number, a theory of factorization, and genera." D. Shanks. In Symposia in Pure Math, volume 20. 1971. doi: 10.1090/pspum/020
- [V+17] "Measuring small subgroup attacks against Diffie-Hellman". Luke Valenta, David Adrian, Antonio Sanso, Shaanan Cohney, Joshua Fried, Marcella Hastings, J. Alex Halderman, Nadia Heninger. www.ndss-symposium.org/ndss2017/ndss-2017-programme/measuring-small-subgroup-attacks-against-diffie-hellman/
- [vOW96] "On Diffie-Hellman key agreement with short exponents." P. C. Van Oorschot and M. J. Wiener. In EUROCRYPT, 1996. doi: 10.1007/3-540-68339-9_29

RSA

Rivest, Shamir, Adleman [RSA78]

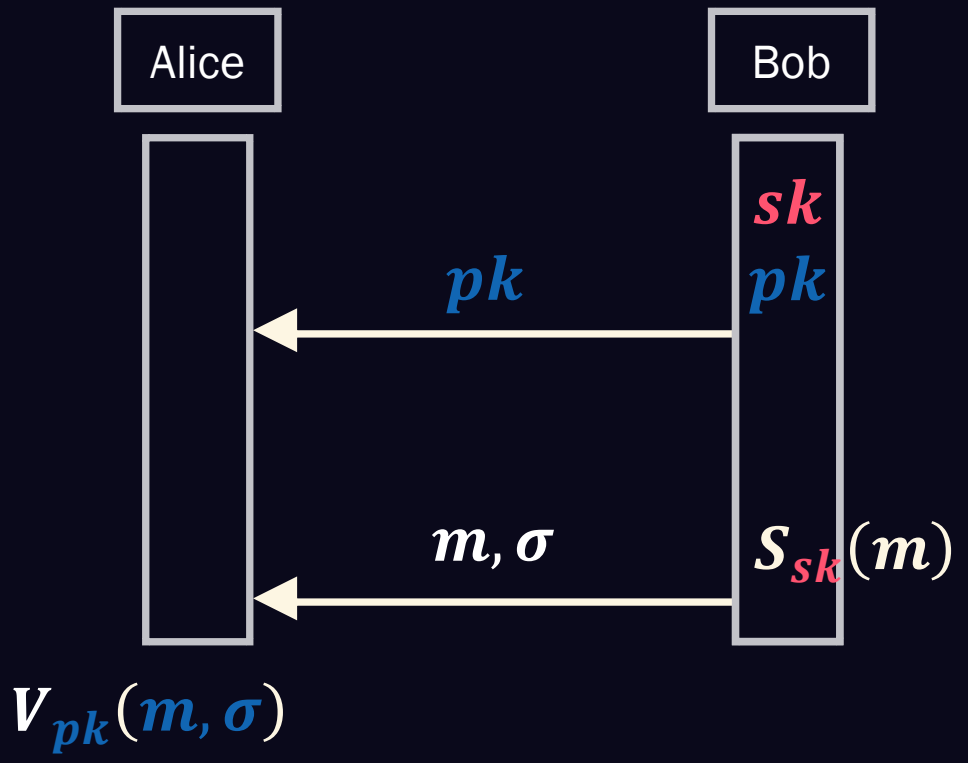
Public key encryption

- Bob generates a (public, secret) key **pair**
- Confidentiality: anyone with pk can **Encrypt** a message m, only Bob with sk can **Decrypt**.



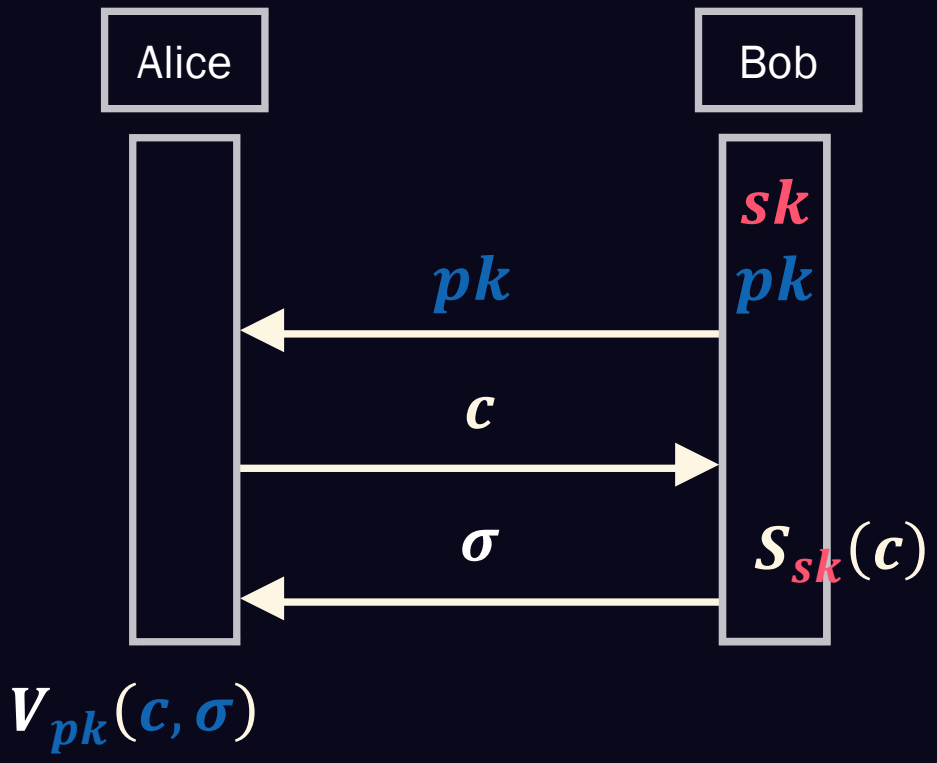
Public key signature

- Bob generates a (public, secret) key **pair**
- Non-repudiation: only Bob with sk can **Sign** a message m; anyone with pk can **Verify** that the signature is authentic.



Public key authentication

- Bob generates a (public, secret) key **pair**
- Authentication: only Bob with sk can **Sign** a challenge r; anyone with pk can **Verify** that the signature is authentic.



RSA

[RSA78]

- It is possible to find numbers d, e, n such that $\boldsymbol{m^{de} \equiv m \pmod n}$
- $\boldsymbol{pk} = (n, e)$, $\boldsymbol{sk} = (n, d)$.
- $\boldsymbol{E_{pk}(m)}$: $c = m^e \pmod n$
- $\boldsymbol{D_{sk}(c)}$: $m = c^d \pmod n$
- $\boldsymbol{S_{sk}(c)}$: $\sigma = c^d \pmod n$
- $\boldsymbol{V_{pk}(\sigma, c)}$: $c \stackrel{?}{=} \sigma^e \pmod n$

RSA

[BS23]

- Choose an integer $l > 2$ and an odd integer $e > 2$, often $\mathrm{e} = 65537$.
- RSA Gen(l, e):
  - *generate a random* l *-bit prime p such that* $\gcd(e, p-1) = 1$
  - *generate a random* l *-bit prime q such that* $\gcd(e, q-1) = 1$ *and* $q \neq p$
  - $n \leftarrow pq$
  - $d \leftarrow e^{-1} \ mod \ (p-1)(q-1)$ *(Euler's* φ*)*
  - *output* (n, d).

RSA

[BS23]

- Choose an integer $l > 2$ and an odd integer $e > 2$, often $\mathrm{e} = 65537$.
- RSA Gen(l , e):
  - *generate a random* l *-bit prime* p *such that* $\gcd(e, p-1) = 1$
  - *generate a random* l *-bit prime* q *such that* $\gcd(e, q-1) = 1$ *and* $q \neq p$
  - $n \leftarrow pq$
  - $d \leftarrow e^{-1} \ mod \ (p-1)(q-1)$ *(Euler's* φ*)*
  - *output* (n, d).
- $\boldsymbol{pk} = (n, e)$, $\boldsymbol{sk} = (n, d)$.

Construction

- For:
  - p, q *prime*
  - $n = pq$
  - a coprime to n
- and Euler's totient function
  - $\phi(n) = (p-1)(q-1)$
- Euler's generalization of Fermat's little theorem:
  - $a^{\phi(n)} \equiv 1 \pmod{n}$

- Choose e, d such that

$$ed = k\phi(n) + 1$$
$$\equiv 1 \pmod{\phi(n)}$$

- So for any m

$$m^{ed} = m^{k\phi(n)+1}$$
$$= m \cdot m^{k\phi(n)}$$
$$\equiv m \pmod{n}$$

- $m^{k\phi(n)} = m^{\phi(n)} \cdot \ldots \cdot m^{\phi(n)}$
$$\equiv 1 \pmod{n}$$

Construction – fixed e

- Public exponent $e = 65537$ (usually); in binary $e = 2^{16} + 1 = 10000000000000001$
- (*Bézout's identity*) Given a, b, integers, there exist x, y such that
$$ax + by = \gcd(a, b)$$
- $\gcd(a, b)$ can be efficiently computed without factoring by the Euclidean Algorithm.
- $\gcd(a, b)$ and x, y can be efficiently computed without factoring by the Extended Euclidean Algorithm.

[MvOV01]#2.4.2

- Suppose e is fixed, such that
$$\gcd\big(e, \phi(n)\big) = 1$$
- this is a prerequisite for the multiplicative inverse to exist. Let $a = e$, $b = \phi(n)$ in the EEA; we then obtain x, y such that
$$ex + \phi(n)y = 1$$
$$ex \equiv 1 \ \big(mod\ \phi(n)\big)$$

References

- [BS23] A Graduate Course in Applied Cryptography. Dan Boneh and Victor Shoup, version 0.6, Jan. 2023. https://toc.cryptobook.us/
- [RSA78] “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems”. Rivest, R.; Shamir, A.; Adleman, L., 1978. Communications of the ACM. 21 (2): 120–126. doi: 10.1145%2F359340.359342

Attacks on RSA

Modulus factoring
Weak keys and messages
Oracles

FONDAZIONE BRUNO KESSLER
CENTER FOR CYBERSECURITY
ALEPH APPLIED CRYPTOGRAPHY

Security against factoring

- p, q must be random primes, factors of $n = qp$ must be “difficult” to find.
- Suppose we could recover p, q. It would be easy to compute $\phi(n)$. By construction, the private exponent is the multiplicative inverse of the public $d = e^{-1} \ (mod\ \phi(n))$. This is easy to compute if you know $\phi(n)$.
- The difficulty of factoring n by the best known algorithm (e.g. NFS) is an upper bound to security. From [SP 800-56b]:

n bit length	2048	3072	4096	6144	8192
max security	112	128	152	176	200

[logjam] (2015): downgrade to export grade cipher suites 512-bit DH

- DH with prime fields and large group orders: most efficient DLog algorithm is NFS. Many parts of the implementations can be shared for factoring.
- “If the NSA has not factored a 1024-bit RSA key, I'm going to be very disappointed and I'm gonna ask where my tax dollars are going”. [logjam] (2015)

Bits	Precomputation for $\mathbb{G}$ (core-years)	~3k cores (2015)	Discrete Log	36 cores (2015)
DH-512	10	~1 week	10 core-min’s	~70 sec’s
RSA-512	0.83			
DH-768	35,000		2 core-days	
RSA-768	900			
DH-1024	45,000,000		30 core-days	
RSA-1024	1,120,000			

Re-used prime: factoring by common factor

- No formula for every possible prime is known. Generating primes is hard: either you choose from a sub-class with a known formula – might be small or vulnerable – or you generate random integers and test for primality – test might be probabilistic or very time-consuming.
- If two moduli n_1, n_2 have a common factor p, it is easy to recover $p = \gcd(n_1, n_2)$ and hence the other factors q_1, q_2.
- Weak keys with a common factor were found in 0.2% certificates and keys by [L12] and 0.5% of TLS certificates in 2012 [HDWH12]
- In [TLS 1.2], the server sends its public key in a TLS certificate during the protocol handshake. The key is used either to provide a signature on the handshake (when Diffie-Hellman key exchange is negotiated) or to encrypt session key material chosen by the client (when RSA-encrypted key exchange is negotiated).

Primes are too close: Fermat factoring

- Suppose, for some small b,
  - $p = a + b$
  - $q = a - b$
- Then
  - $n = a^2 - b^2$
- Any odd integer n can be represented this way (Fermat).
- Find a solution, e.g. start from $a = \sqrt{n}$, increment until $a^2 - n$ is a square.

https://en.wikipedia.org/wiki/Fermat's_factorization_method

$e = 3$: try cube root

- Modular arithmetic is expensive, and many devices are constrained. Suppose e is too small, e.g. 3. Then in probability the ciphertext $c = m^3 << n$ and it will be easy to recover the plaintext as $\sqrt[3]{c}$ (cube root attack).
- If you suspect the ciphertext before mod was only a little larger than n, you can also try a few small multiples $\sqrt[e]{c + kn}$
- A similar argument applies if e is standard but n is ridiculously large.

$e = 3$, 3 signatures: Håstad's broadcast

- Suppose the same message is encrypted to three different receivers:
  - $c_i \equiv m^3 (mod\ n_i)$
- By the Chinese Remainder Theorem, since n_i are coprime, there exists a unique r
  - $r \equiv c_i\ (mod\ n_i)$
  - $\equiv m^3\ (mod\ n_1 n_2 n_3)$
- and we can find it using e.g. Gauss' algorithm. Since both $r, m < \min(n_i)$,
  - $m = \sqrt[3]{r}$

$e = 3$, 3 signatures: Håstad's broadcast

- Suppose the same message is encrypted to three different receivers:
  - $c_i \equiv m^3 (mod\ n_i)$
- By the Chinese Remainder Theorem, since n_i are **coprime**, there exists a unique r
  - $r \equiv c_i\ (mod\ n_i)$
  - $\equiv m^3\ (mod\ n_1 n_2 n_3)$

If they are not, they have a common factor. Try that first.

- and we can find it using e.g. Gauss' algorithm. Since both $r, m < \min(n_i)$,
  - $m = \sqrt[3]{r}$
- Works for any feasibly small e (e.g. 17), not just 3.
- Works for different e and deterministic padding of m, in general.

Chinese Remainder Theorem (CRT)

[MvOV01]

- (CRT) If the integers n_i are pairwise relatively prime, then the system of simultaneous congruences

$$x \equiv a_i \pmod{n_i}$$

has a unique solution x modulo $n = \prod n_i$.

Fact 2.120

- *(Gauss' algorithm)* The unique solution may be computed as

$$x = \sum_i N_i \left(N_i^{-1} \bmod n_i \right)$$
$$N_i = n/n_i$$

Fact 2.121

- More efficient algorithms exist.

§14.5

Malleability

- RSA is malleable: given some ciphertext $c = p^e$, for any chosen s we may craft
  - $c' = c \cdot s^e \pmod n$
  - $= (ps)^e \pmod n$,
- which is also a valid ciphertext.

Decryption oracle: exploit malleability

- Suppose you have an encrypted secret $c = s^e$.
- Suppose an oracle decrypts any ciphertext for you and returns the decryption, iff it does not contain s.
- Choose some other g:
  - *Send modified ciphertext* $C = g^e c$
  - *Receive decrypted* $m = C^d = g^{ed} c^d = gs$
- Since g is known, compute g^{-1} to retrieve s.
- This challenge is academic but could be instanced with server logs of failed decryption dumps, and is a simple example of the following attack.

Parity / LSB oracle

- Suppose an oracle takes as input a ciphertext c and responds
  - *whether the decrypted plaintext* $m = c^d$ *is even or odd, i.e.*
  - *whether the LSB of* m *is* 0 *or* 1*, i.e.*
  - *the oracle returns* $m \mod 2$.
- Exploit malleability: send $c' = c2^e$; the oracle decrypts $(2m)^{ed} \mod n = 2m \mod n$. Note $2m$ will always be even, and n will always be odd.
  - *Case* $2m > n$. *Then*
    - $2m \pmod n$ will be odd, and
    - $(2m \pmod n) \mod 2$ will be 1.
  - *Case* $2m < n$. *Then*
    - $2m \pmod n$ will be even, and
    - $(2m \pmod n) \mod 2$ will be 0.

Iteration

- Every bit of information improves our knowledge of the interval $m \in [L, U]$
- Sending $(2m)^e$, we learn from the oracle's reply:
  - $\texttt{0}$: $0 < m < n/2$
  - $\texttt{1}$: $n/2 < m < n$
- Next, send $(4m)^e$. We now have 4 cases, depending on 2 replies:
  - $\texttt{0,0}$: $0 < m < n/4$
  - $\texttt{0,1}$: $n/4 < m < n/2$
  - $\texttt{1,0}$: $n/2 < m < 3n/4$
  - $\texttt{1,1}$: $3n/4 < m < n$

Iteration

- Every bit of information improves our knowledge of the interval $m \in [L, U]$
- For each bit learned:
  - 0: decrease upper bound

$$U = (U + L)/2$$

  - 1: increase lower bound

$$L = (U + L)/2$$

Padding

- The modulus is an upper bound on the size of the plaintext. A 2048-bit modulus (byte length $l_n = 256$) cannot be used to represent a ciphertext larger than 256 bytes.
  If the message is of a shorter length $l_m < l_n$, we need padding. Padding also helps prevent m being too small to protect against e^{th} root.
- [PKCS#1 v1.5] generate a padding string ps of random but non-zero bytes, of length $l_{ps} = l_n - l_m - 3$ (no less than 8).
  The encryption block will be

- OAEP [PKCS#1 v2.2] roughly speaking uses a hash function instead of random bytes.

Padding oracle

Bleichenbacher [B98]

- Recall RSA is malleable: an attacker can craft $c' = c \cdot s^e \ mod \ n = (ps)^e \ mod \ n$.
- [B98] Suppose there exists an oracle to tell whether a ciphertext is correctly padded with PKCS#1 v1.5, i.e. the decrypted ciphertext starts with `0x0002`. An attacker can try new s until the oracle informs that

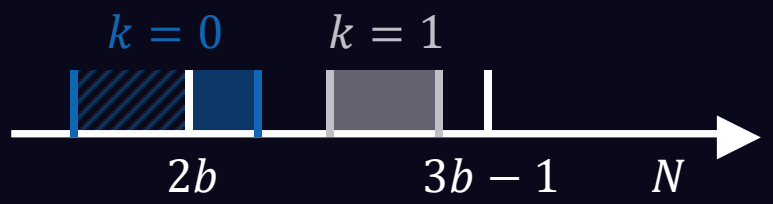
$$2b \leq ms \ \ (mod \ n) < 3b$$

- with $b = 2^{8(l_n - 2)}$. Keep changing s until you find m – full strategy in [B98].
- A 1024-bit RSA key should provide ≈ 80 bits of security. The [B98] oracle attack is estimated to succeed in $\approx 2^{20}$ operations.

$b = 2^{8(l_n - 2)}$	00 01 00 00 00 00 00 …
$2b$	00 02 00 00 00 00 00 …
$3b - 1$	00 02 FF FF FF FF FF …

Interval intersection

- $2b \leq ms \pmod N \leq 3b - 1$
- $2b \leq ms - kN \leq 3b - 1$ for some $k \in \mathbb{Z}$
- $\frac{(2b+kN)}{s} \leq m \leq \frac{(3b-1+kN)}{s}$



- $s_{i+1} = 2s_i$

Padding oracle

Bleichenbacher [B98]

- A 1024-bit RSA key should provide ≈ 80 bits of security. The [B98] oracle attack is estimated to succeed in $\approx 2^{20}$ operations.
- [DROWN] found Bleichenbacher oracles in servers still enabling SSLv2 connections with export-grade restrictions in 2016.
- [ROBOT] found oracles available in TLS 1.2 servers in 2017.

Other attacks

- Pollard $p-1$ weak keys: the modulus has one prime factor p for which $p-1$ is B-smooth for some low value of B. These keys can be broken with Pollard's p−1 attack.
- Wiener weak keys: with small private exponent $d < \frac{1}{3}n^{1/4}$, which can be broken with an implementation of Wiener's attack;
- [ROCA] weak keys: products of primes of the form $p = kM + (65537^a \bmod M)$ for some known M and unknown but small integers k, a. Can be identified and broken using the Coppersmith method.
- CTFTime challenges on Coppersmith, Wiener. See also http://www.science.unitn.it/~sala/cybersecuritymonth2014/

References

- [B98] "Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1". Bleichenbacher, D. 1998. url: archiv.infsec.ethz.ch/education/fs08/secsem/bleichenbacher98.pdf
- [BS23] A Graduate Course in Applied Cryptography. Dan Boneh and Victor Shoup, version 0.6, Jan. 2023. https://toc.cryptobook.us/
- [DROWN] "DROWN: Breaking TLS using SSLv2". Aviram, N., Schinzel, S., et al. 2016. url: drownattack.com
- [FIPS 198-1] "Advanced Encryption Standard (AES)" doi: doi.org/10.6028/NIST.FIPS.198-1. url: csrc.nist.gov/publications/detail/fips/198/1/final

References

- [HDWH12] “Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices”. Heninger, Durumeric, Wustrow, Halderman, USENIX 2012. url: www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/heninger, https://factorable.net/weakkeys12.extended.pdf
- [L+12] “Ron was wrong, Whit is right”. Lenstra, A. et al. 2012. url: ia.cr/2012/064
- [MvOV01] “Handbook of Applied Cryptography”, cacr.uwaterloo.ca/hac/
- [MD5] “The MD5 Message-Digest Algorithm”. RFC 1321 url: tools.ietf.org/html/rfc1321

References

- [PKCS#1 v1.5] RFC 2313. url: tools.ietf.org/html/rfc2313
- [PKCS#1 v2.2] RFC 8017. url: tools.ietf.org/html/rfc8017
- [ROBOT] "Return Of Bleichenbacher's Oracle Threat" Böck, H. Somorovsky, J. and Young, C. 2018 url: robotattack.org
- [ROCA] "The Return of Coppersmith's Attack: Practical Factorization of Widely Used RSA Moduli". Nemec, Sys, Svenda, Klinec, Matyas ACM CCS 17. doi: 10.1145/3133956.3133969. url: https://acmccs.github.io/papers/p1631-nemecA.pdf

References

- [RSA78] "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". Rivest, R.; Shamir, A.; Adleman, L., 1978. Communications of the ACM. 21 (2): 120–126. doi: 10.1145%2F359340.359342
- [SP 800-38b] "Recommendation for Block Cipher Modes of Operation: the CMAC Mode for Authentication". doi: 10.6028/NIST.SP.800-38B. url: csrc.nist.gov/publications/detail/sp/800-38b/final
- [SP 800-56b] "Recommendation for Pair-Wise Key-Establishment Using Integer Factorization Cryptography" revision 2, 2019. doi: 10.6028/NIST.SP.800-56Br2. url: csrc.nist.gov/publications/detail/sp/800-56b/rev-2/final
- [TLS 1.2] "The Transport Layer Security (TLS) Protocol Version 1.2". RFC 5246. url: tools.ietf.org/html/rfc5246

Digital Signatures

Digital Signature Standard (DSS) [FIPS 186]: RSA, DSA, ECDSA

Forgeries and simple attacks

FONDAZIONE BRUNO KESSLER
CENTER FOR CYBERSECURITY
ALEPH APPLIED CRYPTOGRAPHY

Signature forgery

- **Existential**: create **at least one** message-signature pair with a valid signature not created by the holder of the private key
- **Selective**: chosen but fixed message
- **Universal**: any message
- Plain/textbook RSA is
  - *existentially forgeable under known message attack*
  - *universally forgeable under chosen-message attack*

Weak message forgery

- Some signatures are independent of the private key:
  - $m = \mathbf{0}$: $m^d = \mathbf{0} \ \forall d > 0$
  - $m = \mathbf{1}$: $m^d = \mathbf{1} \ \forall d$
  - $m = \boldsymbol{n} - \mathbf{1}$: $m^d = \boldsymbol{n} - \mathbf{1} \ \forall d$

Multiplicative forgery

- Suppose a signer has already generated
  - $\sigma_1 = m_1^d$ *for some* m_1
  - $\sigma_2 = m_2^d$ *for some* m_2
- $\sigma_1\sigma_2 = (m_1 m_2)^d = \sigma(m_1 m_2)\ mod\ n$
- Multiplicative homomorphism: $\prod\sigma(m) = \sigma(\prod m)$.

Homomorphism

An encryption function E is homomorphic with respect to the function f if there is an efficiently computable function g such that

$$g\big(E(m)\big) = E\big(f(m)\big)$$

This property allows to compute on data without breaking confidentiality of data or result.

Blind signature

- RSA is malleable: given some ciphertext $c = m^e$, for any chosen r we may craft
  - $c' = c \cdot r^e \pmod n$
  - $= (mr)^e \pmod n$,
- which is also a valid ciphertext.
- (Blind signature) Suppose Alice wants a signature of a message m, without revealing m to the signer Bob.
  - *Alice draws a random r, sends $r^e p$*
  - *Bob replies with a signature $\sigma(r^e m) = (r^e m)^d = rm^d \pmod n$*
  - $r^{-1} rm \pmod n$ *is a valid signature for m, which the signer has never "seen".*

Signatures and hash functions

- In practice, messages are far too long to be signed
- Hash-then-Sign the message digest $h = H(m)$:
  - $\sigma(m) = \big(H(m)\big)^d$
- Assumes finding collisions is infeasible!
- In [hashclash], chosen-prefix collisions in MD5 were used to craft a CA certificate (RSA-signed).
- In [SLOTH], collisions in MD5 were used to break TLS 1.2 client authentication (RSA-signed handshake transcript).

Digital Signature Algorithm (DSA)

- Parameter generation
- Key generation
- Signature
- Verification

Digital Signature Algorithm (DSA)

- **Parameter generation:** (H, p, q, f)
- Key generation
- Signature
- Verification

Public algorithm parameters:

- A hash function, H
- a random prime p (~ key length)
- a random prime q (~ digest size) divisor of $p - 1$
- some $f \in [2, p-2]$, usually 2
- $g = f^{(p-1)/q} \ mod \ p$, a generator of a subgroup of order q in the multiplicative group of GF(p)

Digital Signature Algorithm (DSA)

- Parameter generation: (H, p, q, f)
- **Key generation:** $(\boldsymbol{d}, \boldsymbol{e} = \boldsymbol{g^d})$
- Signature
- Verification

Private key: a random $d \in [1, q-1]$.

Public key: $e = g^d \mod p$.

Digital Signature Algorithm (DSA)

- Parameter generation: (H, p, q, f)
- Key generation: $(d, e = g^d)$
- **Signature:** (r, s)
  - $r = (g^k \bmod p) \bmod q$
  - $s = (h + dr)k^{-1} \bmod q$
- Verification

Signature of a message m:

- Compute the hash $h = H(m)$, keep the left-most bits to the bit length of q
- Choose **random secret** $k \in [2, q - 1]$
- $r = (g^k \bmod p) \bmod q$ *
- $s = (h + dr)k^{-1} \bmod q$ *

* If = 0, try different k.

Digital Signature Algorithm (DSA)

- Parameter generation: (H, p, q, f)
- Key generation: $(d, e = g^d)$
- Signature: (r, s)
  - $r = \left(g^k \mod p\right) \mod q$
  - $s = (h + dr)k^{-1} \mod q$
- **Verification:**
  - $\boldsymbol{r = \left(e^{rs_i} g^{hs_i} \bmod p\right) \bmod q}$

Verification of a signature (r, s) for message m:

(r, s) is valid for $h = H(m)$ iff

$$r = \left(e^{rs_i} g^{hs_i} \mod p\right) \mod q$$

where $s_i = s^{-1} \mod q$

Attack on k: private key recovery

- $s = (h + dr)k^{-1}$. (h, r, s) are public, (d, k) are secret. k must be **random and different for every signature**.
- Suppose k could be recovered; then
  - $\boldsymbol{d = (sk - h)/r}$

Attack on k: private key recovery

- $s = (h + dr)k^{-1}$. (h, r, s) are public, (d, k) are secret. k must be **random and different for every signature**.
- Suppose k could be recovered; then
  - $\boldsymbol{d = (sk - h)/r}$
- Suppose the same $\boldsymbol{k}$ were used to sign two different digests, h, h'. Easy to tell as the public signatures have the same r. Then we have two equations, two unknowns:
  - $\begin{cases} s = (h + dr)k^{-1} \\ s' = (h' + dr)k^{-1} \end{cases}$
  - $k(s - s') = (h - h')$
  - $k = (h - h')(s - s')^{-1}$
- Similarly for **related $\boldsymbol{k}$**, e.g. $k + 1$ (see here) or partial knowledge of k (here).

Elliptic Curve Digital Signature Algorithm (ECDSA)

- Parameter generation: (H, E, G, q)
- Key generation: $(d \sim Z_q, Q = d \times G)$
- Signature: $(r, s), k \sim Z_q$
  - $(x, y) = k \times G$
  - $r = x \mod q$ *
  - $s = (h + dr)k^{-1} \mod q$ *
- Verification: (r, s) is valid for m iff
  - $r = x' \mod q$
    - where $(x', y') = a \times G + b \times Q$
    - $a = hs^{-1} \mod q$
    - $b = rs^{-1} \mod q$

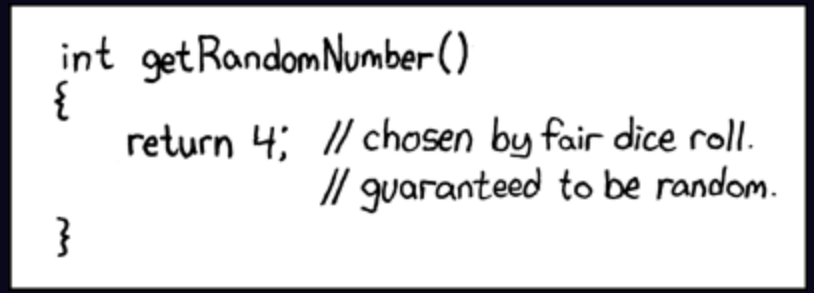
- Public algorithm parameters:
  - *an elliptic curve E*
  - *G: base point of E - generates a subgroup of large prime order q*

$h = H(m)$, keep the left-most bits to the bit length of q

* If = 0, try different random k.

In real life: bugs

- (EC)DSA: same attacks on (same or related) k
  - *[fail0verflow10] Sony PS3: compromised code signing key*
  - *Bitcoin tx signature: compromised wallet*
  - *[BH19] compute hundreds of Bitcoin private keys and dozens of Ethereum, Ripple, SSH, and HTTPS private keys, from biased nonces (short k, common prefix, common suffix)*



https://xkcd.com/221/

In real life: bugs

- (EC)DSA: same attacks on (same or related) k
- ECDSA: $(r = \mathbf{0}, s = \mathbf{0})$
  - *[CVE-2022-21449] "Easily exploitable vulnerability allows unauthenticated attacker with network access via multiple protocols to compromise Oracle Java SE, Oracle GraalVM Enterprise Edition. Successful attacks of this vulnerability can result in unauthorized creation, deletion or modification access to critical data or all Oracle Java SE, Oracle GraalVM Enterprise Edition accessible data. Note: This vulnerability applies to Java deployments, typically in clients running sandboxed Java Web Start applications or sandboxed Java applets, that load and run untrusted code (e.g., code that comes from the internet) and rely on the Java sandbox for security. This vulnerability can also be exploited by using APIs in the specified Component, e.g., through a web service which supplies data to the APIs."*

In real life: bugs

- (EC)DSA: same attacks on (same or related) k
- ECDSA:
  - *Verification: signature* (r, s) *is valid for* m *iff*
    - $r = x' \mod q$
      - *where* $(x', y') = a \times G + b \times Q$
      - $a = hs^{-1} \mod q$
      - $b = rs^{-1} \mod q$
  - s^{-1} *computed by Fermat's little theorem:*
    - $s^{n-1} \equiv 1 \pmod n$
    - $s^{n-2} \equiv s^{n-1} \pmod n$

In real life: bugs

- (EC)DSA: same attacks on (same or related) k
- ECDSA: $(r = \mathbf{0}, s = \mathbf{0})$
  - *Verification:* (r, s) *is valid for* m *iff*
    - $\mathbf{r} = x' \mod q$
      - *where* $(x', y') = a \times G + b \times Q$
      - $\boldsymbol{a} = hs^{-1} \mod q$
      - $\boldsymbol{b} = rs^{-1} \mod q$
  - s^{-1} *computed by Fermat's little theorem:*
    - $s^{n-1} \equiv 1 \pmod n$
    - $\mathbf{0}^{n-2} \equiv s^{-1} \pmod n$

References

- [BH19] “Biased Nonce Sense: Lattice Attacks against Weak ECDSA Signatures in Cryptocurrencies.” Joachim Breitner and Nadia Heninger. Financial Cryptography and Data Security 2019, https://fc19.ifca.ai/preproceedings/104-preproceedings.pdf
- [CVE-2022-21449] https://nvd.nist.gov/vuln/detail/CVE-2022-21449, https://neilmadden.blog/2022/04/19/psychic-signatures-in-java/
- [fail0verflow10] “Console hacking 2010: PS3 epic fail”. bushing, marcan, segher, sven. 27th Chaos Communication Congress, https://fahrplan.events.ccc.de/congress/2010/Fahrplan/events/4087.de.html

References

- [FIPS 186] "Digital Signature Standard (DSS)". doi: 10.6028/NIST.FIPS.186-4. url: csrc.nist.gov/publications/detail/fips/186/4/final
- [hashclash] "Short Chosen-Prefix Collisions for MD5 and the Creation of a Rogue CA Certificate". Stevens, M. et al. Crypto 2009. url: www.win.tue.nl/hashclash/rogue-ca/
- [RFC 6979] "Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)". url: tools.ietf.org/html/rfc6979
- [SLOTH] "Security Losses from Obsolete and Truncated Transcript Hashes". url: www.mitls.org/pages/attacks/SLOTH

Random number generators

FONDAZIONE BRUNO KESSLER
CENTER FOR CYBERSECURITY
ALEPH APPLIED CRYPTOGRAPHY

Random (?) numbers

- Individual numbers are not random.
- Random Variables associate events of uncertain outcome with numbers:

$$X : E \to \mathbb{R}$$

- Example: coin toss
- $X(e) = \begin{cases} 1 & e = \text{heads} \\ 0 & e = \text{tails} \end{cases}$
- Uniformity: $\#E = n$,

$$\mathbb{P}(X = x) = 1/n$$

- Independence:

$$\mathbb{P}(X \mid Y) = \mathbb{P}(X)$$

```
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
               // guaranteed to be random.
}
```

xkcd.com/221/

FONDAZIONE BRUNO KESSLER | CENTER FOR CYBERSECURITY
ALEPH APPLIED CRYPTOGRAPHY

Pseudorandom numbers

- Intuition: autoregressive function of previous states

$$x_{t+1} = g(x_{t,\dots\, t-k})$$

  such that the sequence X_t “looks random”.

- Statistical test suites to determine whether the sequence matches uniform and independent bits: [LS07] [SP 800-22]

dilbert.com/strip/2001-10-25

Pseudorandom numbers

- Intuition: autoregressive function of previous states

$$x_{t+1} = g(x_{t,\dots\, t-k})$$

  such that the sequence X_t “looks random”.
- Statistical test suites to determine whether the sequence matches uniform and independent bits: [LS07] [SP 800-22]
- Functions vs physics:
  - *Quick to compute*
  - *Take up much less space than sensors*
  - *Available on demand*

Pseudorandom number generators

- Examples:
  - $x_{t+1} = ax_t \pmod m$ — *Lehmer*
  - $x_{t+1} = ax_t + b \pmod m$ — *Linear Congruential Generator*
  - $x_{t+i} = ax_{t+k} \oplus \left(({x_t}^L \mid x_{t+1}^R)\, A \right)$ — *Mersenne twister [NMMT98]*
- Internal state (variables in memory)
- Initial "seed" x_0 - must not be easy to guess or influence
- Generate deterministic sequence
- Sequence repeats with finite period

Cryptographically Secure PRNG

- "Next Bit Test"
  - *Given k bits of output, infeasible to predict next bit with probability > 50%*
- State Compromise Extension Resistance:
  - *Given RNG state, infeasible to reconstruct previous numbers.*
- Neither LCG or MT is CS.
- Periodicity is not necessarily the problem:
  - *The Mersenne Twister MT19937 PRNG with 32-bit word length has a periodicity of 2^{19937}-1*
  - *After 624 MT19937 observations, every subsequent value can be predicted.*

LCG

- $x_{t+1} = ax_t + b \pmod m$
- State x_t, multiplier a, increment b, modulus m
- Seed: x_0
- (Hull-Dobell [HD62]): The period is equal to m for all x_0 iff:
  - m, b *are relatively prime*
  - $a - 1$ *is divisible by all prime factors of* m
  - $a - 1$ *is divisible by 4 if* m *is divisible by 4*

LCG state recovery

- Suppose we can collect output from an LCG $x_{t+1} = ax_t + b \pmod m$
- Suppose we know a and m, but not b:

$$b = x_1 - ax_0 \pmod m$$

2 observations

- Suppose we know m, but not a, b:

$$\begin{cases} x_1 = ax_0 + b \pmod m \\ x_2 = ax_1 + b \pmod m \end{cases}$$

$$x_2 - x_1 = a(x_1 - x_0) \pmod m$$

3 observations

- Suppose we don't know the modulus:

$$x_{t+1} = ax_t + b + k_t m$$

every observation adds a new unknown

Recovering the modulus m

- Random multiples of m are likely to have $\gcd = m$.
- Find $f(x) \equiv 0 \ (mod\ m)$, which is equivalent to $f(x) = km$.

Let $\delta_i = x_{i+1} - x_i$

$$\begin{aligned}\delta_1 &= x_2 - x_1 \\ &= ax_1 + b - (ax_0 + b) \\ &= a(x_1 - x_0) \\ &= a\delta_0\end{aligned}$$

Then

$$\begin{aligned}f(x_0, x_1, x_2) &\coloneqq \delta_2 \cdot \delta_0 - \delta_1 \cdot \delta_1 \\ &= (a^2\delta_0 \cdot \delta_0) - (a\delta_0 \cdot a\delta_0) \\ &\equiv 0 \ (mod\ m)\end{aligned}$$

- [B83] prediction algorithm, [J17] worked example, [MvOV01] §5.1 other references – truncation, more linear terms, non-linear functions.

References

- [B83] “Inferring a Sequence Generated by a Linear Congruence”. J Boyar, 1983. doi: 10.1007/978-1-4757-0602-4_32
- [HD62] “Random Number Generators”. Hull, T. E.; Dobell, A. R. SIAM Review. 4 (3): 230–254. doi: 10.1137/1004061.
- [J17] Cracking RNGs: Linear Congruential Generators. J Jedynak, 2017 https://tailcall.net/blog/cracking-randomness-lcgs/
- [LS07] “TestU01: A C Library for Empirical Testing of Random Number Generators”. L'Ecuyer, Simard, ACM Transactions on Mathematical Software 33, 2007. http://simul.iro.umontreal.ca/testu01/tu01.html

References

- [MvOV01] “Handbook of Applied Cryptography”, Menezes, van Oorschot, Vanstone 2001. cacr.uwaterloo.ca/hac/
- [NMMT98] “Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator”. Nishimura, Makoto, Matsumoto, Takuji, ACM Transactions on Modeling and Computer Simulation 8, 1998.
- [SP 800-22] “A Statistical Test Suite for the Validation of Random Number Generators and Pseudo Random Number Generators for Cryptographic Applications”. url: csrc.nist.gov/projects/random-bit-generation/documentation-and-software