IT-Sicherheit

Asymmetrische Kryptographie

Prof. Dr. Dominik Merli, Prof. Dr. Lothar Braun

Sommersemester 2020

Hochschule Augsburg - Fakultät für Informatik

Symmetrisch → Asymmetrisch

Bisher

- Symmetrische Kryptographie
  - Ein geheimer Schlüssel (engl. secret key)
  - Gleicher Schlüssel für Ver-/Entschlüsselung
  - Gleicher Schlüssel für Generierung und Verifikation von MACs
- Vorteile
  - Kompakte Implementierung möglich
  - Hohe Performance möglich
  - Sicherheitseigenschaften sehr gut untersucht, z.B. bei AES
- Nachteil
  - Sender und Empfänger benötigen den gleichen Schlüssel
  - Eigener Schlüssel pro zwei Kommunikationspartner

Neues Thema

- Asymmetrische Kryptographie
  - Ein geheimer Schlüssel (engl. private key)
  - Ein öffentlicher Schlüssel (engl. public key)
  - Verschlüsseln mit Public Key, entschlüsseln mit Private Key
  - Signieren mit Private Key, verifizieren mit Public Key
  - Auch ”Public Key Kryptographie” genannt
- Vorteil
  - Sender und Empfänger benutzen unterschiedliche Schlüssel
  - Einfachere Schlüsselverteilung von öffentlichen Schlüsseln
- Nachteile
  - Implementierungen größer und langsamer als bei symm. Kryptographie
  - Sicherheitseigenschaften komplexer als bei symm. Kryptographie

Anwendungen

- Schlüsselaustausch
- Digitale Signaturen (Integrity, Authenticity, Non-Repudiation)
    - Verträge
    - Firmware
    - E-Mails
    - Zertifikate
    - ...
- Public Key Verschlüsselung (Confidentiality)
    - Schlüsselverschlüsselung
    - E-Mails
    - ...

Geschichte der asymmetrischen Kryptographie I

- **1973/74:** Clifford Cocks entdeckt ”non-secret encryption” (RSA), Malcom Williamson erfindet Schlüsselaustausch (Diffie-Hellman) (erst 1997 vom britischen Geheimdienst veröffentlicht)
- **1976:** Whitfield Diffie and Martin Hellman beschreiben Diffie-Hellman Schlüsselaustausch Protokoll
- **1977:** Ron Rivest, Adi Shamir und Leonard Adleman entwickeln RSA Algorithmus auf Basis des Faktorisierungsproblems
- **1979:** Michael Rabin veröffentlicht Rabin Cryptosystem (Problem bewiesenermaßen so schwer wie Faktorisierung)

Geschichte der asymmetrischen Kryptographie II

- **1985:** Taher Elgamal veröffentlicht ElGamal Cryptosystem (basiert auf Discrete Logarithm Problem)
- **1986/87:** Victor Miller und Neal Koblitz entdecken kryptographische Algortihmen auf Basis von elliptischen Kurven
- **1991:** NIST schlägt Digital Signature Algorithm (DSA) vor
- **2005:** Daniel J. Bernstein schlägt hochperformante und patentfreie elliptische Kurve `Curve29915` vor
- **2016:** NIST sucht öffentlich nach Post-Quantum Algorithmen

RSA

Wozu RSA?

- Entdeckt von
  - Clifford Cocks (GCHQ, 1973)
  - Ron Rivest, Adi Shamir und Leonard Adleman (MIT, 1977)
- Erstes asymmetrisches Verschlüsselungsverfahren
- Erstes Verfahren für digitale Signaturen
- Basiert auf Faktorisierungsproblem
- Immer noch weit verbreiteter Einsatz
- **Hauptsächliche Einsatzzwecke**
  - Verschlüsselung kleiner Datenmengen, z.B. Schlüssel
  - Digitale Signaturen, z.B. E-Mails oder Zertifikate

Faktorisierungsproblem

- Primzahlen
  - Natürliche Zahlen, die größer als 1 und nur durch 1 und sich selbst teilbar sind
  - Alle natürlichen Zahlen lassen sich als Produkt von Primzahlen darstellen

- Faktorisierungsproblem
  - Effizienter Algorithmus zur Berechnung von Primfaktoren einer sehr großen Zahl bisher nicht bekannt

RSA Schlüsselerzeugung

- Schlüssel
  - Public Key $k_{pub} = (n, e)$
  - Private Key $k_{priv} = d$

- Algorithmus
  1) Wähle zwei große Primzahlen p und q (kann in der Praxis sehr lange dauern)
  2) Berechne $n = p \cdot q$
  3) Berechne $\Phi(n) = (p-1)(q-1)$
  4) Wähle den Public Exponent $e \in \{1, 2, ..., \Phi(n) - 1\}$ so dass $gcd(e, \Phi(n)) = 1$
  5) Berechne Private Key d so dass $d \cdot e \equiv 1 \; mod \; \Phi(n)$

$\Phi()$: Eulers Phi Funktion, $gcd()$: Größter gemeinsamer Teiler

RSA Verschlüsselung

- Gegeben
    - Public Key $k_{pub} = (n, e)$
    - Plaintext x
- Verschlüsselung
    - Ciphertext $y = x^e \; mod \; n$
    - wobei $x, y \in \mathbb{Z}_n$

- Gegeben
  - Private Key $k_{priv} = d$
  - Ciphertext y
- Entschlüsselung
  - Plaintext $x = y^d \ mod \ n$
  - wobei $x, y \in \mathbb{Z}_n$

- Gegeben
  - Private Key $k_{priv} = d$
  - Nachricht m
- Signatur-Generierung
  - Signatur $s = m^d \ mod \ n$
  - wobei $s, m \in \mathbb{Z}_n$

RSA Signaturverifikation

- Gegeben
  - Public Key $k_{pub} = (n, e)$
  - Nachricht m
  - Signatur s
- Verifikation
  - $m' = s^e \ mod \ n$
  - wenn $m' \equiv m \ mod \ n$ dann gültig, ansonsten ungültig
  - wobei $m, m', s \in \mathbb{Z}_n$

RSA Schlüssellängen und Sicherheitslevel

- Deutlich längere Schlüssel als symmetrische Algorithmen
- Aktuelle Einschätzung laut NIST SP 800-57 (Jan. 2016):

Sicherheit	RSA Schlüssel
$\leq$ 80-bit	1024-bit
$\leq$ 112-bit	2048-bit
$\leq$ 128-bit	3072-bit
$\leq$ 192-bit	7680-bit
$\leq$ 256-bit	15360-bit

- Schlüssellänge wächst deutlich schneller als Sicherheitsniveau

RSA Implementierungen

- Hauptbestandteil: modulare Exponentiation
  - Benötigt viele Multiplikations-Operationen für große Zahlen
  - Muss für Praxistauglichkeit beschleunigt werden, z.B. mit dem Square-and-Multiply Algorithmus

- Weitere Beschleunigungen
  - Verschlüsselung/Signaturverifikation: kurze öffentliche Exponenten e
  - Entschlüsselung/Signaturgenerierung: Chinese Remainder Theorem (CRT)

Beispiel: OpenSSL

- Performance Test

```
$ openssl speed rsa1024 rsa2048 rsa4096
[...]
                  sign    verify    sign/s verify/s
rsa 1024 bits 0.000101s 0.000007s   9856.0 152752.8
rsa 2048 bits 0.000690s 0.000020s   1449.3  50329.7
rsa 4096 bits 0.004593s 0.000075s    217.7  13309.1
```

- Ergebnisse
  - Intel Core i7-6600U @ 2.60GHz $\rightarrow \sim$ 1500 sign/s (rsa2048)
  - Verifikation deutlich schneller als Signatur-Generierung
  - Doppelte Schlüssellänge $\rightarrow$ 6-7x sign Zeit, 3-4x verify Zeit

Aufgabe

Recherchieren Sie Performance-Ergebnisse für RSA!

Beachten

Schlüssellänge
Operation (Sign/Verify/Encrypt/Decrypt)
Prozessortyp und -takt
Implementierungshinweise/-kommentare

Elliptische Kurven

Was sind elliptische Kurven?

- Elliptische Kurve = mathematisches Konstrukt
  - $y^2 \equiv x^3 + a \cdot x + b \ mod \ p$
  - Punkte auf Kurve: $P = (x_P, y_P)$
- Definiert über Galoiskörper $GF(p)$ oder $GF(2^m)$
- Kurvenoperationen bauen auf Operationen im Galoiskörper auf
  - Punktaddition $R = P + Q$
  - Punktverdoppelung $R = P + P$
  - Skalarmultiplikation $R = d \cdot P$
- Elliptic Curve Cryptography (ECC) basiert auf Discrete Logarithm Problem (DLP)

Vorteil: ECC Schlüssel

- Generierung einfacher als für RSA
- Deutlich kürzer als RSA für gleiche Sicherheit
- Aktuelle Einschätzung laut NIST SP 800-57 (Jan. 2016):

Sicherheit	RSA Schlüssel	ECC Schlüssel
$\leq$ 80-bit	1024-bit	160-bit
$\leq$ 112-bit	2048-bit	224-bit
$\leq$ 128-bit	3072-bit	256-bit
$\leq$ 192-bit	7680-bit	384-bit
$\leq$ 256-bit	15360-bit	512-bit

- ECC Schlüssel wachsen linear mit Sicherheitslevel

Website: Schlüssellängen

- Link: https://www.keylength.com
- Sehr hilfreich bei der Auswahl von Schlüssellängen
- Basiert auf verschiedenen Standards und Empfehlungen

ECDSA Schlüsselgenerierung

- Schlüssel
  - Public Key $k_{pub} = (p, a, b, q, A, B)$
  - Private Key $k_{priv} = d$

- Algorithmus
  1) Wähle elliptische Kurve mit
     - Modulus p
     - Koeffizienten a und b
     - Punkt A, generiert zyklische Gruppe der Ordnung q
  2) Wähle zufällige Zahl d, wobei $0 < d < q$
  3) Berechne $B = d \cdot A$

ECDSA Signatur

- Gegeben
  - Private Key $k_{priv} = d$
  - Nachricht m

- Signatur-Generierung
  1) Wähle zufälligen Ephemeral Key k_E, wobei $0 < k_E < q$
  2) Berechne Punkt $R = k_E \cdot A$
  3) $r = x_R$
  4) Berechne $s = (h(m) + d \cdot r) \cdot k_E^{-1} \bmod q$
  5) Signatur: (r, s)

$h()$: Hash-Funktion

ECDSA Signaturverifikation

- Gegeben
  - Public Key $k_{pub} = (p, a, b, q, A, B)$
  - Nachricht m
  - Signatur (r, s)

- Verifikation
  - $w = s^{-1} \; mod \; q$
  - $u_1 = w \cdot h(m) \; mod \; q$
  - $u_2 = w \cdot r \; mod \; q$
  - Berechne $P = u_1 \cdot A + u_2 \cdot B$
  - wenn $x_P \equiv r \; mod \; q$ dann gültig, ansonsten ungültig

$h()$: Hash-Funktion

ECC Implementierungen

- Meist in vier Schichten aufgeteilt
  - Modulare Arithmetik
    (Addition, Subtraktion, Multiplikation, Inversion)
  - Gruppenoperationen (Punktaddition, Punktverdopplung)
  - Skalarmultiplikation
  - Protokolloperationen (z.B. ECDSA)

- Auch für eingebettete Systeme geeignet
  - Microcontroller- und FPGA-Implementierungen
  - Hardware-Software Co-Design interessant
  - Auch für Chipkarten und sogar RFID-Tags

Beispiel: OpenSSL

- Performance Test

```
$ openssl speed ecdsa
[...]
                              sign    verify    sign/s verify/s
192 bit ecdsa (nistp192)   0.0001s   0.0002s   16189.7   4171.5
224 bit ecdsa (nistp224)   0.0001s   0.0001s   16813.7   8226.2
256 bit ecdsa (nistp256)   0.0000s   0.0001s   26929.9  12396.2
384 bit ecdsa (nistp384)   0.0002s   0.0008s    5137.2   1247.6
521 bit ecdsa (nistp521)   0.0003s   0.0007s    2894.0   1521.5
163 bit ecdsa (nistk163)   0.0002s   0.0004s    5267.1   2382.1
233 bit ecdsa (nistk233)   0.0004s   0.0005s    2617.1   1906.9
```

- Ergebnisse
    - Intel Core i7-6600U @ 2.60GHz $\rightarrow \sim$ 17000 sign/s (nistp224)
    - Signieren mit nistp224 mehr als 10x schneller als mit rsa2048
    - Auch Kurventyp ausschlaggebend, z.B. nistp224 vs. nistk233
    - Doppelte Schlüssellänge $\rightarrow$ 3-4x sign Zeit, 3-4x verify Zeit

Aufgabe

Recherchieren Sie Performance-Ergebnisse für ECDSA!

Beachten

Kurventyp/-bezeichnung
Schlüssellänge
Operation (Sign/Verify)
Prozessortyp und -takt
Implementierungshinweise/-kommentare

Post-Quantum Kryptographie

Angriffe mit Quanten-Computern?

- Entwicklung von Quanten-Computern schreitet voran
- Shor Algorithmus bedroht Public Key Kryptographie (Faktorisierungsproblem und Discrete Logarithm Problem)
- 2019 unternahm Google einen ersten Versuch zur Demonstration der ”Quantenüberlegenheit” auf Basis eines Quantencomputers mit 53 Qubits
- Bisher keine reale Bedrohung, aber Forschung muss jetzt beginnen

PQCrypto Projekt

- Europäisches Forschungsprojekt 2015 – 2018
- Ziel: Erarbeitung von Post-Quantum Algorithmen Portfolio
- Ergebnis: Standardisierung von Post-Quantum Algorithmen ist langfristige Aufgabe
- https://pqcrypto.eu.org/

Initiale PQCrypto Empfehlungen

- **Public Key Verschlüsselung**
  McEliece mit binären Goppa Codes
  (Parameter: n=6960, k=5413, t=119)

- **Public Key Signaturen**
  Hash-basierte Signatur-Funktionen, z.B. XMSS oder SPHINCS-256

NIST Post-Quantum Crypto Ausschreibung

- Ausschreibung startete im Dez. 2016
- Aktuell: Kandidaten der zweiten Runde werden analysiert
- Nach 5-7 Jahren: Entwürfe für Standardisierung
- http://csrc.nist.gov/groups/ST/post-quantum-crypto/

Gibt es noch Fragen?