

# 1 gesammelte mathematische Grundlagen

**quadratischer Rest** Eine Zahl  $a$  ist ein quadratischer Rest bzgl eines Moduls  $m$ , wenn sie zu  $m$  teilerfremd ist und es eine Zahl  $x$  gibt, für die die Kongruenz  $x^2 \equiv a \pmod{m}$  gilt

**quadratischer Nichtrest** Existiert für eine zu  $m$  teilerfremde Zahl  $a$  keine Lösung  $x$  der obigen Kongruenz, dann nennt man  $a$  einen *quadratischen Nichtrest* mod  $m$ . Zu  $m$  nicht teilerfremde Zahlen werden nicht klassifiziert.

**Jacobi-Symbol,  $n$  ist Primzahl**  $\left(\frac{a}{n}\right) = \begin{cases} 1 & \text{wenn } a \text{ ein quadratischer Rest zu } n \text{ ist} \\ -1 & \text{wenn } a \text{ kein quadratischer Rest zu } n \text{ ist} \\ 0 & \text{wenn } a \text{ und } n \text{ nicht teilerfremd sind} \end{cases}$

**Jacobi-Symbol,  $n$  ist keine Primzahl** Sei die Primfaktorzerlegung von  $n = p_1^{v_1} \cdot p_2^{v_2} \cdot \dots \cdot p_k^{v_k}$ , dann ist  $\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{v_1} \cdot \dots \cdot \left(\frac{a}{p_k}\right)^{v_k}$ . Die einzelnen Faktoren können nun nach dem ersten Fall ( $n$  ist Primzahl) behandelt werden. Das Jacobi-Symbol kann auch ohne Kenntnis der Primfaktorzerlegung effizient berechnet werden. Hier ist ein Jacobi-Symbol von  $\neq -1$  eine notwendige Bedingung dafür, dass  $a$  ein quadratischer Rest bzgl.  $n$  ist. Ein Jacobi-Symbol von 1 ist nicht hinreichend.

**Euler-Kriterium zur Berechnung des Legendre-Symbols**  $\left(\frac{a}{p}\right) \equiv a^{\frac{p-1}{2}} \pmod{p}$ ,  $p$  ist Primzahl,  $p \neq 2$ . Für  $p = 2$  gilt  $-1 \equiv 1 \pmod{2}$

**Eulersche  $\varphi$ -Funktion**  $\varphi(n)$  gibt an, wieviele Zahlen  $a$  zu  $n$  teilerfremd sind.  $\varphi(p)$ ,  $p$  Primzahl =  $p - 1$ ,  $\varphi(m \cdot n) = \varphi(m) \cdot \varphi(n)$ , falls  $m, n$  teilerfremd.

**Satz von Euler-Fermat**  $a^{\varphi(m)} \equiv 1 \pmod{m}$  wenn  $a, m$  teilerfremd sind. Hauptsatz für RSA. Hiermit wird bewiesen, dass  $(m^e)^d \pmod{n} = m$  gilt.

## 2 Symmetrische Verfahren

### 2.1 Klassische Verfahren

**Atbash** Alphabet wird von hinten nach vorne benutzt. a wird auf z abgebildet, b auf y, ...

#### Caesar

- Verschlüsselung:  $E_k(a_i) = b_i = (a_i + k) \pmod{26}$
- Entschlüsselung:  $D_k(b_i) = a_i = (b_i - k) \pmod{26}$
- Schlüssel aus  $k \in \{1, \dots, 25\}$  sinnvoll

#### Affines System

- Verschlüsselung:  $E_{a,b} = b_i = (a_i \cdot a + b) \pmod{r}$

#### Keyword Caesar

- Permutation des Alphabets wird generiert durch ein Kennwort und einem Offset  $x$
- Kennwort wird an Position  $x$  geschrieben, dann werden die verbliebenen Buchstaben angehängt (bzw. vor das Kennwort geschrieben)
- Bsp: (video mit Offset 3) xyzvideoabcfghjklmnpqrstuw

## Vigenere

- Kennwort sind  $x$  Zahlen  $k_0, \dots, k_{x-1}$
- Das  $i$ -te Zeichen des Plaintexts wird um  $k_i \bmod x$  Zeichen verschoben
- Bsp:  $E_{1,2}(\text{beispiel}) = \text{cgjuqkfn}$

## Garbage-in-between

- Kennwort ist eine streng monotone Funktion
- $E_f(a_1, \dots, a_n) = b_1, \dots, b_m$  mit  $b_{f(i)} = a_i$ , alle übrigen  $b_j$  enthalten nur Garbage
- $f$  gibt also an, an welchen Positionen im verchlüsselten Text die Buchstaben des Plaintexts sind

## Hill

- Schlüssel ist eine Matrix  $M$  der Dimension  $d \times d$ , die ein Inverses modulo 26 hat
- Verschlüsseln geschieht in Blöcken der Länge  $d$ , indem  $M \cdot (a_1, a_2, \dots, a_d)^T$  berechnet wird
- Entschlüsselung geschieht durch Multiplikation mit der Inversen von  $M$

## Playfair

- Paare von Buchstaben werden verschlüsselt
- 25 Buchstaben ( $i = j$ ) werden in einer  $5 \times 5$ -Matrix angeordnet (Schlüssel)
- Ein Buchstabenpaar bildet nun ein Rechteck in der Matrix, die anderen beiden Ecken entsprechen dem verschlüsselten Buchstabenpaar
- Sonderregelung, falls beiden Buchstaben in der gleichen Zeile/Spalte liegen

## Autoclave und Autoclave'

- Schlüssel hat die Länge  $l$ , Plaintext hat die Länge  $n$
- Die ersten  $l$  Zeichen  $p_i$  des Plaintexts werden jeweils mit dem  $i$ -ten Zeichen  $k_i$  des Schlüssels modulo  $r$  addiert
- Alle anderen Zeichen  $p_j, j > l$  werden mit dem Zeichen  $p_{j-l}$  modulo  $r$  addiert
- Der Schlüssel wird also durch den Plaintext ergänzt
- Beim Autoclave' wird der Schlüssel durch den Crypttext statt durch den Plaintext ergänzt

## Jefferson Wheel

- Besteht aus 36 drehbaren Holzscheiben (feste Hardware)
- Auf jeder Scheibe steht eine Permutation des Alphabets
- Plaintext wird dargestellt, danach werden alle Scheiben um einen Wert  $x$  gedreht. Das ist dann der Crypttext
- Der Schlüssel ist also neben dem Wert  $x$  der Automat selber

## One-Time-Pad

- Einziges beweisbar sicheres Verfahren
- Zu jedem Paar Crypttext-Plaintext gibt es einen passenden Schlüssel
- Ver- und Entschlüsselung läuft über bitweises XOR-rechnen
- Schlüssel muss genauso lang sein, wie Plaintext

## Häufigkeitsanalyse

- Bleiben die statistischen Eigenschaften der Sprache nach dem Verschlüsseln erhalten (monoalphabetisch, ohne Transposition), so kann man die Häufigkeiten analysieren
- In zB der deutschen Sprache treten folgende Buchstaben am häufigsten aus: E, N, I, R, S, A, T
- Analog für Buchstabenpaar, Wörter,...

## Kasiski-Analyse

- Bestimmung der möglichen Schlüssellänge
- Taucht im Crypttext eine Buchstabenfolge doppelt auf, so kann man annehmen, dass das gleiche Wort (zB "der") mit dem selben Teil des Schlüssels verschlüsselt ist
- Die Schlüssellänge ist nun (vorausgesetzt, es war wie angenommen) ein Teiler des Abstandes zwischen den beiden Crypttext-Stellen

## C-36 (M-209) Verfahren

- Erstellung eines möglichst langen Schlüssels
- Beginn ist eine *step-figure*  $H$ , die aus 6 Zeilen der Länge 17, 19, 21, 23, 25 und 26 besteht ( $\in \{0, 1\}$ )
- Diese Zeilen werden nun wiederholt, die erste Periode tritt somit erst beim Spaltenvektor  $v_{101.405.850}$  auf
- Sei  $M$  nun eine  $6 \times 27$ -Matrix mit maximal 2 Einsen pro Spalte
- Die *Hit-number*  $\#_{v_i}$  eines Vektors  $v_i$  der *step figure*  $H$  ist die Anzahl der Elemente  $\geq 1$  aus dem Vektor, der sich aus  $v_i \times M$  ergibt
- $E_{M,H}(a_0, \dots, a_n) = c_0, \dots, c_n$  mit  $c_i = \#_{v_i} - a_i \pmod{27}$
- $D_{M,H}(c_0, \dots, c_n) = a_0, \dots, a_n$  mit  $a_i = \#_{v_i} - c_i \pmod{27}$

## 2.2 DES

- Erstes öffentlich bekanntgegebene Verschlüsselungsverfahren (1978)
- 56 Bit Schlüssellänge
- Blöcke mit 64 Bit werden verschlüsselt
- Grundidee: Nur Eingabe und Ergebnis sind zu sehen, Zwischenergebnisse nicht sichtbar, viele mögliche Wege (Schlüsselraum), Schlüssel zwischendurch änderbar

## Grundlegender Ablauf

- Eingabe ist ein 64 Bit langer Plaintext und ein 64 Bit langer Schlüssel, bei dem auf den Positionen 8,16,24,...,64 Parity-Bits stehen
- Der Schlüssel sowie der Plaintext werden am Anfang direkt anhand einer Tabelle permutiert, der Plaintext wird danach in zwei 32 Bit lange Blöcke geteilt
- Danach folgen 8 Runden, die fast immer gleich ablaufen
- Am Ende wird der verschlüsselte Plaintext nocheinmal anhand einer Tabelle permutiert und als Crypttext ausgegeben (Ich nenne den Eingabetext bis zur Ausgabe Plaintext)
- Die Entschlüsselung wird mit dem gleichen Algorithmus durchgeführt, wobei die einzelnen Rundenschlüssel in umgekehrter Reihenfolge verwendet werden.

## Die 8 Runden

- In den Runden  $K > 1$  wird der Schlüssel anhand einer Tabelle geshiftet, bevor er in die Berechnung der Runde eingreift
- In allen Runden wird der Schlüssel (nach dem Shiften bzw im Fall  $K = 1$  direkt) von 56 auf 48 Bit mit einem festen Verfahren gekürzt
- Pro Runde wird der Plaintext in einen ersten und einen zweiten Block mit je 32 Bit aufgeteilt. Der zweite Block entspricht genau dem ersten Block des Plaintexts der nächsten Runde und er ist die eigentliche Eingabe in die Funktion  $F$ .
- Die Ausgabe der Funktion  $F$  wird mit dem ersten Block der Runde ge-XOR-t, das Ergebnis daraus bildet den zweiten 32 Bit Block der nächsten Runde

## Die Funktion $F$

- $F$  ist eine Funktion, die aus dem 32 Bit langen Block des Plaintexts und dem 48 Bit Block des Schlüssels einen 32 Bit Block erstellt
- Dazu werden die 32 Plaintext-Bits auf 48 Bits aufgebohrt (Tabelle), danach mit den 48 Bit des Schlüssels ge-XOR-t und danach wieder auf 32 Bit abgespeckt (Tabelle)

## Vor- und Nachteile

- Vorteile: sehr schneller Algorithmus, ist erster Standard, Probleme bei der Analyse sind NP-schwer
- Nachteile: konstante Schlüssellänge, Permutations- und Umrechnungstabellen (S-Boxen) sind fest (Vermutung einer Hintertür), vollständige Schlüsselsuche verteilt in 24h möglich

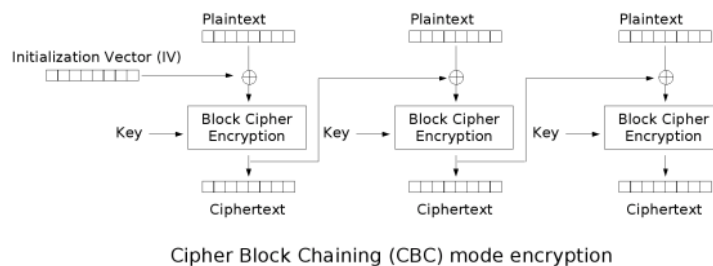
## Verbesserungsmöglichkeiten

- Triple-DES: 3 Schlüssel notwendig, Plaintext  $w$  wird mit  $k_3$  ver-, das Ergebnis danach mit  $k_2$  ent- und das Ergebnis davon danach mit  $k_1$  verschlüsselt
- Multi-DES: 3 Schlüssel notwendig, Plaintext wird in 3 Blöcke aufgeteilt, jeder Block wird mit einem eigenen Schlüssel verschlüsselt und die Ergebnisse werden danach in der korrekten Reihenfolge konkateniert
- DESX:  $E_{K,K_1,K_2}^{DESX}(w) = K_2 \oplus E_K^{DES}(K_1 \oplus (w))$ ,  $K$  ist 56 Bit lang,  $K_1$  und  $K_2$  sind 64 Bit lang, "sicher" vor vollständiger Schlüsselsuche
- komplizierte Mischung aus den oberen Verfahren

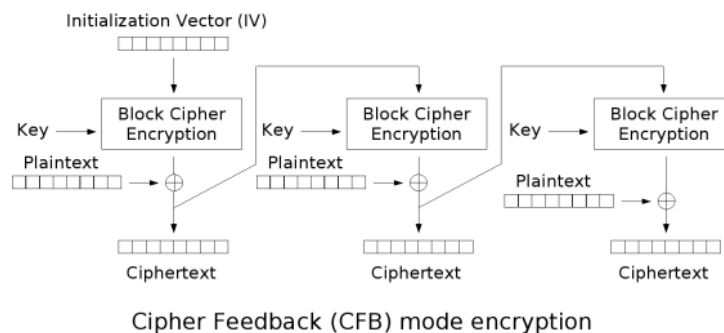
## 2.3 Operationsmodi

**Electronic Codebook Mode (ecb)** Plaintextblöcke werden nacheinander und unabhängig voneinander mit dem selben Schlüssel verschlüsselt. Verfahren ist sehr einfach aber auch sehr unsicher. Vorhandene Muster im Text werden nicht verwischt.

**Cipher-Block Chaining Mode (cbc)** Die Plaintextblöcke werden auch hier nacheinander verschlüsselt, jedoch werden sie vorher mit der Crypttext-Ausgabe des letzten Block ge-XOR-t. Für Block 1 wird der Plaintext mit einem zufällig gewählten String ge-XOR-t (da kein vorheriger Block vorhanden ist). Vorhandene Muster werden zerstört. Durch eine fehlerhafte Übertragung eines Blocks kann dieser und der darauf folgende nicht korrekt entschlüsselt werden.



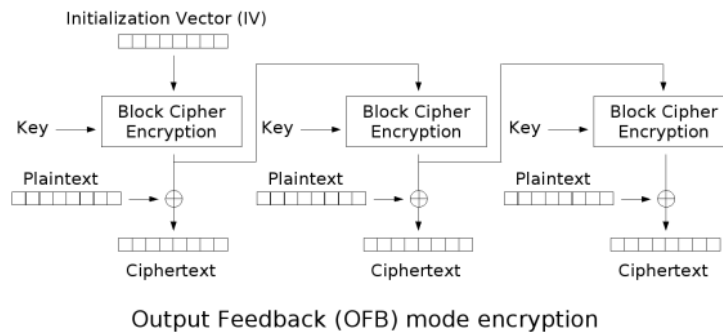
**Cipher Feedback Mode (cfb)** Dieser Modus ist etwas anders als die vorangegangenen: Es werden hier jeweils nicht die Plaintext-Blöcke verschlüsselt, sondern die Crypttext-Blöcke des jeweils vorangegangenen Blocks (Block 1 benutzt wieder einen zufällig gewählten String). Bevor der Crypttext eines Blocks ausgegeben bzw an den nächsten Block weitergegeben wird, wird der Plaintext noch mit dem verschlüsselten String ge-XOR-t (erst XOR-en, dann weitergeben). Dieser Modus ergibt eine Stromchiffre, die den Nachteil hat, dass ein Übertragungsfehler in einem Block wieder diesen und den folgenden zu grossen Teilen zerstört.



**Output Feedback Mode (ofb)** Im großen und Ganzen funktioniert dieser Modus wie der vorherige, bis auf dass der Plaintext erst mit dem verschlüsselten Text ge-XOR-t wird, nachdem dieser an den nächsten Block geleitet wurde (erst weitergeben, dann XOR-en). Fehler wirken sich hier bei weitem nicht so schlimm aus, es wird nur das falsch übertragene Bit fehlerhaft dargestellt.

## 2.4 IDEA

- Ist patentiert, kann also nicht frei verwendet werden
- 128 Bit Schlüssel, verschlüsselt Plaintext-Blöcke mit je 64 Bit, rechnet intern mit 16 Bit Blöcken
- Grundidee ist wildes Mischen von 3 Operationen, die keine algebraischen Gemeinsamkeiten haben: XOR, Addition mod  $2^{16}$  und Multiplikation mod  $2^{16} + 1$



- IDEA besteht aus 8 vollständigen und einer Extrarunde. Für jede Runde  $i \in \{1, 8\}$  werden die Teilschlüssel  $Z_1^{(i)}$  bis  $Z_6^{(i)}$  benötigt, die Extrarunde benötigt die Teilschlüssel  $Z_1^{(9)}$  bis  $Z_4^{(9)}$

### Teilschlüsselgenerierung

Für IDEA werden 52 Teilschlüssel der Länge 16 Bit benötigt

1. Teile den 128 Bit Schlüssel in 8 Teile mit je 16 Bit und benenne diese  $Z_1^{(1)}, Z_2^{(1)}, Z_3^{(1)}, Z_4^{(1)}, Z_5^{(1)}, Z_6^{(1)}, Z_1^{(2)}, Z_2^{(2)}$
2. Shifte die Zeichen des Schlüssels um 25 Bit nach links
3. Teile den dann gewonnenen 128 Bit Schlüssel in 8 Teile mit je 16 Bit und benenne diese  $Z_3^{(2)}, Z_4^{(2)}, Z_5^{(2)}, Z_6^{(2)}, Z_1^{(3)}, Z_2^{(3)}, Z_3^{(3)}, Z_4^{(3)}$
4. Wiederhole dies bis zum 52ten Teilschlüssel,  $Z_4^{(9)}$

### Verschlüsselung

1. Vor der ersten Runde wird der Plaintext in 16 Bit große Stücke aufgeteilt. Diese bilden die 4 Eingangsblöcke der ersten Runde.
2. Die Ausgabe jeder Runde sind auch wieder 4 Blöcke mit je 16 Bit. Diese werden direkt an die nächste Runde weitergeleitet.
3. Die 4 Plaintextblöcke werden nun mit Hilfe der zugehörigen 6 Schlüssel in jeder Runde mit den 3 oben genannten Operationen verschlüsselt.
4. In der Extrarunde werden nur 4 Schlüssel benötigt, diese unterscheiden sich deshalb von den übrigen Runden.
5. Ausgabe sind wieder diese 4 Blöcke mit je 16 Bit

### Entschlüsselung

Der Algorithmus läuft exakt so, wie bei der Verschlüsselung. Der einzige Unterschied ist der, dass der Schlüssel umgeformt verwendet wird:

Aus  $Z_1^{(i)}, Z_2^{(i)}, Z_3^{(i)}, Z_4^{(i)}, Z_5^{(i)}, Z_6^{(i)}$  wird  $Z_1^{(10-i)^{-1}}, -Z_2^{(10-i)}, -Z_3^{(10-i)}, Z_4^{(10-i)^{-1}}, Z_5^{(9-i)}, Z_6^{(9-i)}$  für  $i \in \{1, 8\}$  und  
 aus  $Z_1^{(9)}, Z_2^{(9)}, Z_3^{(9)}, Z_4^{(9)}$  wird  $Z_1^{(1)^{-1}}, -Z_2^{(1)}, -Z_3^{(1)}, Z_4^{(1)^{-1}}$

## 2.5 AES

- Nachfolger von DES
- 128 Bit lange Blöcke und 128, 192 oder 256 Bit lange Schlüssel
- Rijndael (Gewinner des AES-Wettbewerbs) kann jedoch Blöcke und Schlüssel mit 128, 160, 192, 224 oder 256 Bit Länge verarbeiten

- leicht zu implementieren (Referenzcode hatte weniger als 500 Zeilen C-Code)
- überdurchschnittliche Performance
- keine Methode der Kryptoanalyse zum Brechen von AES bekannt
- Intern wird mit 8 Bit Blöcken (1 Byte) gerechnet

### Ablauf von AES

- Jeder Plaintextblock (128 Bit) wird in eine zweidimensionale Tabelle (genannt *State*) mit 4 Zeilen geschrieben, pro Zelle 1 Byte (bei AES also 4 Spalten, bei Rijndael entsprechend mehr)
- Auf jeden Block werden nun nacheinander Teile des erweiterten Originalschlüssels angewendet
- Die Anzahl der Runden  $r$  von AES hängt nur von der Schlüssellänge ab (bei 128 Bit 10 Runden, bei 192 Bit 12 Runden, bei 256 Bit 14 Runden; bei Rijndael hängt die Rundenzahl auch von der Blocklänge ab, die ist bei AES aber konstant)

### Struktur von AES

- Rijndael (State, CipherKey)
  - KeyExpansion(CipherKey,ExpandedKey);
  - AddRoundKey (State,ExpandedKey[0]);
  - for ( $i := 1; i \leq r; i++$ ) Round (State, ExpandedKey[i]);
  - FinalRound (State,ExpandedKey[r]);
- Round (State,ExpandedKey[i])
  - SubBytes (State);
  - ShiftRows (State);
  - MixColumns (State);
  - AddRoundKey (State,ExpandedKey[i]);
- FinalRound (State,ExpandedKey[r])
  - SubBytes (State);
  - ShiftRows (State);
  - AddRoundKey (State,ExpandedKey[r]);

**KeyExpansion** Die Rundenschlüssel müssen die selbe Länge haben, wie die Blöcke. Der Schlüssel muss also auf die Länge  $Blocklänge \cdot (r + 1)$  expandiert werden. Das Ganze geschieht rekursiv. Der Schlüssel wird, wie der Plaintext auch, in eine zweidimensionale Tabelle mit 4 Zeilen und 8 Bit je Zelle geschrieben. Ein weiteres Byte wird nun berechnet, indem ein bestimmtes anderes Byte durch eine S-Box gejagt wird und danach mit einem anderen Byte ge-XOR-t wird. Je nachdem, an welcher Position das zu berechnende Byte liegt, wird es noch mit einem Eintrag aus einer anderen Tabelle per XOR verknüpft.

**AddRoundKey** Der aktuelle Rundenschlüssel wird mit XOR mit dem aktuellen Block verknüpft.

**SubBytes** Jedes Byte wird durch eine S-Box substituiert.

**ShiftRows** Zeile  $i$  wird zyklisch nach links verschoben. Anzahl der Positionen, um die verschoben wird, ist zeilen- und blockabhängig.

**MixColumns** Die einzelnen Spalten (betrachtet als Vektoren) werden mit einer festgelegten Matrix multipliziert. Das Ergebnis ist wieder ein  $1 \times 4$ -Vektor, dessen Einträge alle von jedem Eintrag des Ausgangsvektors abhängen.

## 2.6 Moderne Angriffe

- Known Plaintext, auch Teilinformationen
- Belauschen der Peripherie
- Spyware
- Aufbau von Gleichungssystemen über die Verfahren
- Nutzung von Parallelrechnern, Spezialrechner
- Brute-Force Angriffe
- Fehler in der Implementation
- Hintertüren in Programmen
- Bestechung / unvorsichtiger Anwender
- Timing-Attacks, Rückschlüsse über die Antwortzeiten
- Rückschlüsse über elektrische Felder
- Rückschlüsse über Geräuschentwicklung der CPU

## 3 Public-Key

### 3.1 Einleitung

#### ALLGEMEIN

- bel. Schlüssellänge
- langsamer als klassische Verfahren
- **Unterschrift**
- Einsatz in Protokollen

#### Telefonbuch

- Plaintext: OTTO
- Suche willk. Nummer zu Namen der mit O, T, T, O beginnt
- Entschlüsseln: Durchsuchen des Telefonbuchs und notieren des zugehörigen Anfangsbuchstabens
- mit *inversen* Telefonbuch einfach.
- inverses Telefonbuch ist geheimer Schlüssel.

## Multiplikation & Division

Ann. keiner kann Dividieren:

- $\text{rnd } x$
- $\text{rnd } y$
- $n = x \cdot y$
- $x$  privkey
- $y$  und  $n$  pubkey
- aber: keine Hintertür!
- Es gilt:  $x \cdot y = n$

Verschlüsseln von  $m$ :

- $\text{rnd } z$
- Verschlüsseln:  $c = n \cdot z + m$
- aber: Empf. kann damit nicht entschlüsseln
- sende zusätzlich:  $d = z * y$

Entschlüsseln:

- $m = c - n \cdot z$
- $m = c - x \cdot y \cdot z$
- $m = c - x \cdot d$

- Empfänger veröffentlicht  $E_k$
- $D_{k'}$  ist nur ihm bekannt, nicht eff. aus  $E_k$  berechenbar.
- Sender in gleicher Situation wie Lauscher
- *Knacken* möglich (Man in the middle, social engineering)

Benötigt:

- Suchen von *One-Way-Funktion*
- noch keine Funktion *bewiesene* One-Way-Funktion, aber Kandidaten: Primfaktorzerlegung...

### GRUNDIDEE DER ÜBERTRAGUNG

**B:** Erstelle public ( $E_{k_b}^{PKS}$ ) und private ( $D_{k_b, k'_b}^{PKS}$ ) key

**B→A:** übertrage/veröffentliche public key

**A  $w$ :** verschlüssel  $w$ :  $c = E_{k_b}^{PKS}(w)$

**A→B:** übertrage Krypttext  $c$

**B:** entschlüssel  $c$ :  $w' = D_{k_b, k'_b}^{PKS}(c)$

- Angriff: Teste für alle  $w''$ , ob gilt:  $E_{k_b}^{PKS}(w'') = c$
- Darum muß der Nachrichtenraum für  $w$  groß sein!

### Verschlüsselung: Multiplikation und Division

**B:** Erzeugt  $x, y, n = y \cdot x$

**B→A:** übertrage pubkey  $n$  und  $y$

**A  $w, y, n$ :** Erzeugt  $z$ , bestimme  $c = z \cdot n + w, d = y \cdot z$

**A→B:** Übertrage Krypttext  $c$  und Entschlüsselungshilfe  $d$

**B:** Bestimme  $w' = c - d \cdot x$

## GRUNDIDEE EINER UNTERSCHRIFT

**B**  $m \rightarrow$  **A**: B erzeugt pub ( $E_{k_b}^{PKS}$ )/priv ( $D_{k_b, k'_b}^{PKS}$ ) key und veröffentlich pubkey ( $k_b, E_{k_b}^{PKS}$ )

**B**  $m$ : Bestimmt Signatur  $s = D_{k_b, k'_b}^{PKS}(m)$  (Entschlüsselung von  $m$  mittels privkey)

**B**  $\rightarrow$  **A**: Übertrage  $m, s$  (Nachricht + Signatur)

**A**  $m, s$ : Teste  $m = E_{k_b}^{PKS}(s)$ .  $m$  gleich Verschlüsselung von  $s$  mit pubkey?

Schlüsselraum für  $k'_b$  muß groß sein.

### Unterschrift: Multiplikation und Division

**B**  $m \rightarrow$  **A**: B erzeugt pub  $x$ /priv  $n$  key und veröffentlich pub key ( $n = x \cdot y$ ) und Hilfe ( $y$ )

**B**:  $m, x, y, n$  Erzeuge Zufallszahl  $z$ , bestimme  $d = x \cdot z, s = m - n \cdot z$

**B**  $\rightarrow$  **A**: Übertrage Hilfe  $d$ , Signatur  $s, z$  und Nachricht  $m$

**A**  $m, y, n, s, d, z$ : Teste  $m = y \cdot d + s$  ( $m = E_y(s)$ ) und  $n \cdot z = y \cdot d$  ( $d$  richtig?)

## GRUNDLAGE VON VERSCHLIESSBAREN KÄSTEN

**B**  $m \rightarrow$  **A**: veröffentlich pubkey

**B**  $m \rightarrow$  **A**: bestimmt Krypttext  $s$  (Kasten) von  $m$  (Inhalt) und überträgt diese.

**A**  $s$ : Speichert  $s$  für später.

**B**  $\rightarrow$  **A**: überträgt  $m$  und privkey

**A**: teste  $m = D_{k_b, k'_b}^{PKS}(s)$ : Ist  $s$  Verschlüsselung von  $m$

### verschießbare Kästen: Multiplikation und Division

**B**  $m \rightarrow$  **A**: erzeuge  $x, y, n = y \cdot x$  und übertrage  $y, n$

**B**: bestimme  $c = m - x$ , Verschlüsselung von  $m$  (Box)

**A**  $y, n, m$ : Speichert  $c$

**B**  $\rightarrow$  **A**: übertrage privkey  $x$

**A**:  $y, n, m, x$  Teste  $n = x \cdot y$  ( $n$  richtig?) und entschlüssel  $c$  ( $m = c + x$ )

## ALLGEMEINER AUFBAU

1. Wähle schweres Problem  $P$ , z.B. aus NPC.
2. Wähle leichtes Teilproblem  $P_L$  von  $P$
3. Verändere  $P_L$  zu  $P_S$ , so dass  $P_S$  schwer erscheint
4. veröffentliche  $P_S$  mit Beschreibung zur Verschlüsselung, dabei soll  $P_S$  wie  $P$  erscheinen.

## MÖGLICHE GRUNDLAGEN

- NP-schwere Probleme (SAT, Rucksack)
- Faktorisierung (Primfaktorzerlegung)
- Quadratfreiheit (ist  $n = x \cdot p^2$ )
- Quadratischer Rest (ist  $x^2 \equiv a \pmod n$ )
- diskreter Logarithmus (ist  $a^x \equiv b \pmod n$ )
- elliptische Kurven
- Polynomringe
- fehlertolerante Codes
- Homomorphismen (Aufbau noch unklar)
- endliche Automaten (China)

**Systemaufbau: SAT** Konstruiert werden zwei Boolesche Formeln  $F_0, F_1$  über  $x_1, \dots, x_n, y_1, \dots, y_n$ , mit:

- $F_i : \{0, 1\}^{2n} \rightarrow \{0, 1\}$  für  $i \in \{0, 1\}$  (Belegung von  $F_0, F_1$ )
- $\exists e_1, \dots, e_n \in \{0, 1\}^n$  mit:  $\forall e'_1, \dots, e'_n \in \{0, 1\}^n : F_i(e_1, \dots, e_n, e'_1, \dots, e'_n) = i$  (Es ex. eine Belegung des ersten Teils, so das die Belegung des zweiten Teils beliebig sein kann und  $F_i = i$  gilt.)
- privkey:  $e_1, \dots, e_n$
- pubkey:  $F_0, F_1$
- Es wird immer nur ein Bit übertragen!

**Verschlüsselung:** Um  $m \in \{0, 1\}$  zu übertragen wird  $F_m$  wie folgt umgeformt:

1. Wähle zufällig Belegung  $e'_1, \dots, e'_n$  der Variablen  $y_1, \dots, y_n$
2.  $F'_m : \{0, 1\}^n \rightarrow \{0, 1\}; F'_m(x_1, \dots, x_n) = F_m(x_1, \dots, x_n, e'_1, \dots, e'_n)$
3. Verändere nun  $F'_m$  zu  $F''_m$  wobei  $\forall x_1, \dots, x_n : F''_m(x_1, \dots, x_n) = F'_m(x_1, \dots, x_n)$
4. veröffentliche nun  $F''_m$

**Entschlüsseln:** Aus  $F''_m$  ergibt sich das geheime Bit  $m$  durch einsetzen des privkey's:  $m = F''_m(e_1, \dots, e_n)$ .

Anmerkung.

- Nicht besonders effizient, da für jedes Bit eine vollständige Boolesche Formel übertragen werden muß.
- Aus dem Verfahren folgt, dass aus jedem NP-vollständigem Problem ein PK-Verfahren aufgebaut werden kann (bekannte Reduktionen auf das SAT-Problems).

### 3.2 Rucksackproblem

**Definition** (Rucksackproblem/Knapsack).

**Eingabe**  $r_1, r_2, \dots, r_k, s \in \mathbb{N}$

**Frage**  $\exists I \subset \{1, 2, \dots, k\} : \sum_{i \in I} r_i = s$

**Ausgabe**  $I \subset \{1, 2, \dots, k\} : \sum_{i \in I} r_i = s$

In  $2^k$  Schritten durch Testen jeder Teilmenge lösbar:  $\forall I \subset \{1, 2, \dots, k\}$  teste ob  $\sum_{i \in I} r_i = s$  gilt.

**Verschlüsselung:**

- $E_{r_1, \dots, r_k}^{KS1} : \mathbb{Z}_2^{n \cdot k} \rightarrow \mathbb{Z}^n$
- $D_{r_1, \dots, r_k, x}^{KS1} : \mathbb{Z}^n \rightarrow \mathbb{Z}_2^{n \cdot k}$
- $E_{r_1, \dots, r_k}^{KS1}(a_1, \dots, a_{n \cdot k}) = KS_{r_1, \dots, r_k}(a_1, \dots, a_k) E_{r_1, \dots, r_k}^{KS1}(a_{k+1}, \dots, a_{n \cdot k})$
- $KS_{r_1, \dots, r_k}(a_1, \dots, a_k) = \sum_{1 \leq i \leq k} r_i \cdot a_i$
- Beispiel:  $E_{1,3,5,7}^{KS1}(0110 \ 1101 \ 0001 \ 1001) = 8, 11, 7, 8$
- One-Way-Funktion vorhanden.
- Verfahren unsicher wenn Eingabe wenige Einsen oder Nullen enthält.
- Die Entschlüsselung ist ggf. nicht eindeutig.
- Entschlüsselung (legal und illegal) ist schwer.

## Verbesserungen

1.

**Definition** (Stark steigender Vektor). Ein Vektor  $R = (r_1, r_2, \dots, r_k)$  heißt *stark steigend* wenn gilt:  $\forall j : 1 < j < k : r_j > \sum_{1 \leq i < j} r_i$

$R(1, 2, 4, 13, 30, \dots)$  ist stark steigender Vektor. ( $4 > 1 + 2; 13 > 1 + 2 + 4; 30 > 1 + 2 + 4 + 13$  usw.)

Benutze stark steigende Vektoren zur Verschlüsselung. Damit ist die Analyse aber einfach, auch deshalb, weil der Empfänger bisher keine geheime Informationen hat. *Lösung*: Nicht die stark steigende Version von  $R$  veröffentlichen!

2.

**Definition** (Starke modulare Multiplikation).  $A = (a_1, \dots, a_n)$  stark steigender Vektor,  $m > \max A$ ,  $t : \text{ggT}(m, t) = 1$ , d.h.:  $\exists u : u \cdot t \equiv 1 \pmod m$ . Dann entsteht  $B = (b_1, \dots, b_n)$  aus  $A$  durch *modulare Multiplikation*:  $b_i = t \cdot a_i \pmod m$ . Falls weiterhin gilt:  $m > \sum_{i=1}^k a_i$  entsteht  $B$  aus  $A$  durch *starke modulare Multiplikation*.

*Beachte*. Falls  $B$  aus  $A$  durch mod. Multiplikation entsteht, so auch  $A$  aus  $B$ , dies gilt nicht für die starke mod. Multiplikation.

## Aufbau eines neuen PK-Systems:

- $A = (a_1, \dots, a_k)$  stark steigender Vektor
- wähle  $m > \sum_{i=1}^k a_i$
- bestimme  $t \mid \text{ggT}(t, m) = 1$
- bestimme  $B = (b_1, \dots, b_k) \mid b_i = t \cdot a_i \pmod m$
- veröffentliche  $B$
- Verschlüsselung:  $E_B^{KS2} : \mathbb{Z}_2^{n \cdot k} \rightarrow \mathbb{Z}^n$
- Entschlüsselung:  $D_{A,t,m}^{KS2} : \mathbb{Z}^n \rightarrow \mathbb{Z}_2^{n \cdot k}$

**Lemma 1.**  $A = (a_1, \dots, a_k)$  stark steigender Vektor und  $B$  entsteht aus  $A$  durch starke modulare Multiplikation mittels  $m, t$ .  $u \equiv t^{-1} \pmod m$  ( $u \pmod m = t^{-1}$ ).  $\beta$  beliebig und  $\alpha = u\beta \pmod m \Rightarrow$ :

1. Das KS-Problem  $(A, \alpha)$  ist eindeutig lösbar in Linearzeit. (keine oder eine Lösung, falls Parameter frei wählbar).
2. KSP  $(B, \beta)$  hat keine oder genau eine Lösung.
3. Wenn  $(B, \beta)$  eine Lösung hat, dann ist der Lösungsvektor gleich der eindeutigen Lösung von  $(A, \alpha)$ .

## Sicherheitsaspekte:

- Falls  $k < C \Rightarrow$  Lösbarkeit in  $2^C$  Schritten möglich (für konstantes  $C$ )
- Falls  $\max B < C \Rightarrow$  Lösbarkeit in  $O(1)$
- Übliche Verfahren: halte  $k$  variable
- Die Größe der  $a_i$  ergibt sich dann aus  $k$ :  $a_i$  enthält  $dk - 1 - k + i$  Bits
- **Beispiel**  $k = 100, d = 2, m$  hat 200 Bits  $a_1, \dots, a_k$  : 100 bis 199 Bits (stark steigend)
- Problem: Der stark steigende Vektor  $A$  ist eine starke Einschränkung
- Ein beliebiges  $m', t', A'$  kann zur Analyse dienen.

**Entschlüsseln von KS2: Idee:** suche bel.  $u, m$  welches aus  $B$  stark steigenden Vektor  $A$  macht.  $f_{b_i}(u, m) = u \cdot b_i \pmod m = a_i$ . Beachte:  $a_1$  ist im Vergleich zu  $m$  sehr klein. **Problem:** Es gibt zwei Variablen  $u$  und  $m$

**Idee:** Nomiere die Funktion mit  $m$ , d.h. teile durch  $m$  und bestimme nicht  $u$  und  $m$ , sondern  $\frac{u}{m}$ .  $a_1/m$  ist immer noch klein im Vergleich zu 1. Analoge Überlegungen kann man für  $a_2, a_3, \dots$  machen. Man erhält also nur wenige, sehr kleine Lösungsintervalle, die allen  $a_i$  gemeinsam sind.

- Bestimme nun diese Lösungsbereiche für  $\frac{u}{m}$  und betrachte jeden davon.
- Formuliere Bedingungen an  $\frac{u}{m}$ , die notwendig sind, damit  $A$  stark steigend wird.
- Falls Bereich gef. wird, in dem  $\frac{u}{m}$  eine mögliche Lösung ist, reduziert sich das Problem auf das Suchen *Diophantischer Zahlen* innerhalb dieses Bereiches.
- Folgende zwei Schritte sind notwendig:
  1. Bestimmung der Häufungspunkte
  2. Testen dieser auf mögliche Lösung

#### Entschlüsseln von KS2 (Verfahren):

- Koordinate des  $p$ -ten Min. von  $b_1$  ist nun  $\frac{p}{b_1}$
- Bestimme ob Min. von  $b_2$  nahe an einem Min. von  $b_1$  liegt.
- $-\epsilon < \frac{p}{b_1} - \frac{q}{b_2} < \epsilon$ , für  $1 \leq p \leq b_1 - 1 \wedge 1 \leq q \leq b_2 - 1$
- $\Rightarrow -\delta < b_2 p - b_1 q < \delta$  für passendes  $\delta$
- Betrachte nun diese Ungleichung für die ersten  $s$  Kurven
- Bestimme maximal  $r$  Kandidaten
- $s, r$  sind also *frei* wählbare Parameter
- Wähle  $\delta < \sqrt{\frac{b_1}{2}}$ , Fehlerwahrscheinlichkeit  $(\frac{2}{r})^{s-1}$

$p$  sei nun die zu testende Zahl (also  $p$  Min. der  $b_1$ -Kurve)

- Teile Intervall  $(\frac{p}{b_1}, \dots, \frac{p+1}{b_1})$  in Teilintervalle ohne Sprünge  $(x_j, x_{j+1}) \mid 1 \leq j \leq y$ .
- Auf diesem Intervall haben die Kurven die Gestalt:  $b_i - c_i^j$ .  $c_i^j$  sind also Konstanten die von  $i, j$  abhängen.
- Teste diese Kurven auf starke Steigung und starke modulare Multiplikation. Untersuche also für  $v (= u/m)$  mit  $x_j < v < x_{j+1}$ :  $\sum_{i=1}^k (b_i v - c_i^j) < 1$  starke modulare Multiplikation; und  $\sum_{i=1}^{i-1} (b_i v - c_i^j) < b_i v - c_i^j, 2 \leq i \leq k$
- Falls eine Lösung vorhanden ist, bestimme  $x \in I \cap \mathbb{Q}$

### 3.3 RSA

#### Aufbau

- Wähle Primzahlen  $p, q$
- Setze  $n = p \cdot q, \varphi(n) = (p - 1)(q - 1)$
- Private Key: Wähle  $d > 1$  mit  $\text{ggT}(d, \varphi(n)) = 1$  (sonst existiert kein Inverses)
- Public Key: Berechne  $e$  mit  $e \cdot d \equiv 1 \pmod{\varphi(n)}$  (Inverses zu  $d$ , z.B. durch modulare Invertierung)
- Siehe auch beim Satz von Euler-Fermat in Kapitel 1.

#### VERSCHLÜSSELUNG:

$$E(m) = m^e \pmod{n}$$

#### ENTSCHLÜSSELUNG:

$$D(c) = c^d = (m^e)^d = m^{ed} = m^1 \pmod{n}$$

#### Wichtig:

- $p, q$  sollten nicht nah beieinander liegen
- $p, q$  sollten zufällig gewählt werden (nicht aus Listen wählen)
- $\text{ggT}(p - 1, q - 1)$  sollte klein sein
- $\frac{p-1}{2}$  und  $\frac{q-1}{2}$  sollten auch Primzahlen sein
- $e, d$  sollten gross sein

#### EINDEUTIGKEIT DER ENTSCHLÜSSELUNG

Die Entschlüsselung von RSA ist eindeutig. Plaintext  $w$ , Crypttext  $c = w^e$ ,  $w$  und  $n$  sind teilerfremd

$$w^{\varphi(n)} \equiv 1 \pmod{n} \text{ (Satz von Euler-Fermat)}$$

$$w^{j \cdot \varphi(n)} \equiv 1 \pmod{n}$$

$$w^{ed-1} \equiv 1 \pmod{n}$$

$$w^{ed} \equiv w \pmod{n}$$

$$c^d \equiv w \pmod{n}$$

## Sicherheit RSA

### SICHERHEITSASPEKT VON $\varphi(n)$

Lemma: Wenn man  $\varphi(n)$  berechnen kann, kann man  $p$  und  $q$  berechnen.

Beweis:

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

$$p - q =$$

$$\sqrt{(p-q)^2} =$$

$$\sqrt{p^2 + q^2 - 2n} =$$

$$\sqrt{2n + p^2 + q^2 - 4n} =$$

$$\sqrt{(p+q)^2 - 4n}$$

$$\frac{n - \varphi(n) + 1 + \sqrt{(n - \varphi(n) + 1)^2 - 4n}}{2} = \frac{p+q+p-q}{2} = p$$

### SICHERHEIT VON $d$

Lemma: Falls  $d$  bestimmt werden kann, so können  $p$  und  $q$  probabilistisch bestimmt werden.

1. Wähle  $w$  mit  $1 \leq w \leq n$
2. Versuche,  $n$  mit  $w$  zu faktorisieren (Chance: 50%)
3. Wahrscheinlichkeit nach  $k$  Tests kein passendes  $w$  gefunden zu haben:  $1 - 2^{-k}$

Hier fehlt dann noch das Verfahren, das über 6 Folien geht...

### SICHERHEIT VON $d$

Lemma:

HIER FEHLT NOCH EINIGES ZUR SICHERHEIT VON  $d$  UND  $\varphi(d)$  UND ZU ORAKEL UND BITSICHERHEIT VON RSA

### 3.4 Rabin

#### Aufbau

- Wähle Primzahlen  $p, q$  mit  $p, q \equiv 3 \pmod{4}$
- Teste hierfür Zahlen der Form  $4k + 3$  auf Primzahlen
- $n := p \cdot q$
- Private Key:  $p, q$
- Public Key:  $n$

#### VERSCHLÜSSELUNG:

$$c = E(m) = m^2 \pmod{n}$$

#### ENTSCHLÜSSELUNG:

1. Berechne  $m_p = c^{\frac{p+1}{4}} \pmod{p}$  und  $m_q = c^{\frac{q+1}{4}} \pmod{q}$
2. Berechne mit dem erweiterten euklidischen Algorithmus  $y_p, y_q$  mit  $y_p p + y_q q = 1$
3. Berechne  $r = (y_p p m_p) + (y_q q m_q)$  und  $s = (y_p p m_p) - (y_q q m_q)$
4. Die Kandidaten für  $m$  sind  $\pm r$  und  $\pm s$

Zum Entschlüsseln wird der chinesische Restklassensatz verwendet. Demnach existiert eine Funktion  $\varphi : \mathbb{Z}_n \rightarrow \mathbb{Z}_p \times \mathbb{Z}_q$  mit  $c \rightarrow (c \pmod{p}, c \pmod{q})$  mit dem der Schlüsseltext zerlegt werden kann. Dadurch lassen sich die 2 bzw. 4 Wurzeln modulo  $p$  bzw.  $q$  bestimmen, die dann Kandidaten für den Klartext sind.

Um diesen eindeutig von den anderen Wurzeln unterscheiden zu können, kann man zB. der Nachricht einen eindeutigen Header voranstellen.

#### Sicherheit von Rabin

Das Verfahren von Rabin ist so sicher wie die Faktorisierung schwer ist.

Falls man auf den Algorithmus  $D_{p,q}^{\text{Rabin}}$  zugreifen kann, kann man auch  $n$  faktorisieren:

1. Wähle  $m$  mit  $0 < m < n$
2. Setze  $c := m^2 \pmod{n}$
3. Setze  $y := D_{p,q}^{\text{Rabin}}(c)$
4. Falls  $m \not\equiv \pm y \pmod{n}$  kann  $n$  faktorisiert werden.

## Unterschriften mit Rabin

Um Unterschriften mit Rabin zu ermöglichen, wird eine Hashfunktion  $h$  benötigt.

Erzeugen der Unterschrift:

- Suche zufällig gewähltes  $x$ , für das  $h(m, x)$  ein Quadrat in  $\mathbb{Z}$  ist.
- Bestimme Wurzel  $y$  von  $h(m, x)$
- Die Unterschrift ist dann  $(m, x, y)$
- Verifizieren:  $h(m, x) \equiv y^2 \pmod{n}$  testen

## 3.5 ElGamal

**Einleitung:** Erinnerungen:

- $G$  ist zyklische Gruppe:  $\exists g : G = \{g^m | m \in \mathbb{N}\}$  und dessen Generator wird *primitive Wurzel* genannt.
- Falls  $p$  Primzahl, dann ist  $\mathbb{Z}_p^*$  zyklische Gruppe mit  $\varphi(p-1)$  Generatoren.
- $\mathbb{Z}_p^*$  zyklisch gdw.  $n \in \{1, 2, 4, p^k, 2 \cdot p^k\}$
- $x \in \mathbb{Z}_p^*$  ist Generator gdw.  $x^{(p-1)/q} \neq 1$  für alle Primzahlen  $q$ , die  $p-1$  teilen.
- Da  $\text{ord}(x)$  das  $p-1$  teilt gilt:  $x^{(p-1)/q} = 1 \pmod{p-1}$  oder  $\text{ord}(x) = p-1$

Siehe Beispiel *Keiner Kann Dividieren* in 3.1

ElGamal:

- Schweres Problem: diskreter Logarithmus. Also schwer für gegebene  $a, b, p$  ein  $x$  zu bestimmen mit  $a^x = b \pmod{p}$
- $a \cdot b$  aus dem Beispiel wird dann zu  $a^b \pmod{p}$  und  $a + b$  zu  $a \cdot b \pmod{p}$

**Aufbau:**

- Bestimme große Primzahl  $p$ , so daß  $p-1$  einen großen Primfaktor hat: dafür wähle zuerst große Primzahl  $q$  und teste dann als Kandidaten für  $p$  Zahlen der Form  $2kq + 1$ .
- Bestimme Generator  $g \in \mathbb{Z}_p^*$ , um das zu testen muß  $k$  faktorisiert sein, also wähle  $q$  groß und  $k$  relativ klein.
- Für  $g$  muß für alle Teiler  $q'$  von  $p-1$  gelten  $g^{(p-1)/q'} \not\equiv 1 \pmod{p}$ , damit läßt sich einfach testen ob  $g$  Generator ist.
- Wähle dann zufällig  $x \in \{1, \dots, p-2\}$
- Bestimme  $y = g^x \pmod{p}$
- priv Key ist dann  $(p, g, x)$
- pub Key ist dann  $(p, g, y)$

**Verschlüsseln:** Hierbei wird keine Hintertür durch das System vorgegeben, sondern vom Verschlüsseler eine zusätzliche Information angegeben, mit deren Hilfe nur der Empfänger entschlüsseln kann.

- Verschlüsselungsfkt:  $E_{p,q,y}^{\text{ElGamal}} : \mathbb{Z}_p^* \rightarrow \mathbb{Z}_p^* \times \mathbb{Z}_p^*$
- Wähle  $k$  zufällig mit  $\text{ggT}(k, p-1) = 1$  und:  $a \equiv g^k \pmod{p}$  und  $b \equiv my^k \equiv mg^{xk} \pmod{p}$
- $E_{p,q,y}^{\text{ElGamal}}(m) \rightarrow (a, b)$ , damit hat der Krypttext die doppelte Größe vom Plaintext.

**Entschlüsseln:**

- Entschlüsselt wird mittels der Eigenschaft:  $m = b/a^x \pmod p$
- Entschlüsselungsfkt. ist dann:  $E_{p,q,y}^{ElGamal} : \mathbb{Z}_p^* \times \mathbb{Z}_p^* \rightarrow \mathbb{Z}_p^*$
- Empfang von  $(a, b) = (g^k, my^k) = (g^k, mg^{xk})$ , dann bestimme:
  - $h := a^x \pmod p : a^x \equiv g^{kx} \pmod p$
  - $m := b \cdot h^{-1} \pmod p : b/a^x \equiv y^k m / a^x \equiv g^{kx} m / a^x \equiv m \cdot g^{kx} / g^{kx} \equiv m \pmod p$
- Damit gilt:  $E_{p,q,y}^{ElGamal}(a, b) \rightarrow b/a^x \pmod p$

## ÜBERBLICK

**A**  $x, y, g \rightarrow$  **B** Wähle Primzahl  $p$  mit  $p - 1$  hat großen Primfaktor. Wähle  $g$  und  $x \in \{1, \dots, p - 2\}$ . Bestimme  $y := g^x \pmod p$ . Übertrage dann  $y, g$

**B**  $m, y, g \rightarrow$  **A** Wähle  $k$  mit  $\text{ggT}(k, p - 1) = 1$  und Übertrage  $a \equiv g^k \pmod p$  und  $b \equiv my^k \equiv mg^{xk} \pmod p$

**A** Entschlüssel  $m = b/a^x \pmod p$

## Sicherheitsaspekte:

- ElGamal ist bezüglich Sicherheit und Geschwindigkeit vergleichbar mit RSA
- Sicherheit von ElGamal: Falls  $g^x$  und  $g^k$  gegeben, so ist es schwer  $g^{xk}$  oder  $g^{-xk}$  zu berechnen oder  $m$  wie oben angegeben. Zugrundeliegendes Problem ist als Diffie-Hellman-Problem bekannt.
- Bekannt: Wenn man Problem des diskreten Logarithmus lösen kann, dann auch Diffie-Hellman. Ob Rückrichtung gilt ist offen!

**Unterschrift mit ElGamal:** Systemaufbei wie bei *keiner kann Dividieren* Unterschrift!

## Unterschreiben von $m$

1. Wähle  $k$  zufällig, mit  $1 \leq k \leq p - 2 : \text{ggT}(k, p - 1) = 1$
2. Bestimme Hilfe  $r := g^k \pmod p$  und Signatur  $s := k^{-1}(m - rx) \pmod p - 1$
3. Die Unterschrift ist dann  $(m, r, s)$

## Testen von Unterschrift

1. Teste  $1 \leq r \leq p - 1$
2. Bestimme  $v := g^m \pmod p$  und  $w := y^r r^s \pmod p$
3. Teste  $v \stackrel{?}{=} w$

## UNTERSCHRIFT MIT ELGAMAL

**B**  $m, x, y, g \rightarrow$  **A** Wähle Primzahl  $p$  mit  $p - 1$  hat großen Primfaktor, wähle  $g$  und  $x \in \{1, \dots, p - 2\}$ . Übertrage  $y := g^x \pmod p$  und  $g$ .

**B**  $m, x, y, g \rightarrow$  **A** Wähle  $k$  mit  $\text{ggT}(k, p - 1) = 1$  und Übertrage Unterschrift  $(r, s, m)$ :  $r \equiv g^k \pmod p$ ,  $s := k^{-1}(m - rx) \pmod p - 1$  und  $m$

**A**  $m, y, g$  Teste  $1 \leq r \leq p - 1$ . Bestimme  $v := g^m \pmod p$ ;  $w := y^r r^s \pmod p$ . Teste  $v \stackrel{?}{=} w$

Bei korrekter Unterschrift gilt  $v = w$ , da  $w \equiv y^r r^s \equiv (g^x)^r (g^k)^s \equiv g^{rx} g^{k^{-1}(m - rx)} \equiv g^m \equiv v \pmod p : g^{p-1} \equiv 1 \pmod p$

## Sicherheitsaspekte der Unterschrift mit ElGamal:

- Wenn man disk. Log. eff. berechnen kann, dann kann man auch die Unterschrift brechen

- Zu geg.  $m, r$  ein  $s$  zu bestimmen mit  $g^m = y^r r^s$  ist äquivalent dazu den disk. Log. zu berechnen.
- Es ist offen wie schwer das Fälschen einer Unterschrift ist: Also zu geg.  $m$  die Werte  $r, s : g^m = y^r r^s$  zu bestimmen.
- Falls man  $k$  zu einer Nachricht  $m$  bestimmen kann, so kann man auch priv. key  $x$  bestimmen:  $rx = (m - sk) \pmod{p-1}$ . Da mit hoher Wahrscheinlichkeit  $\text{ggT}(r, p-1) = 1$  kann man  $x$  bestimmen.  $\Rightarrow$  Wichtig guten Zufalls-generator zu verwenden!
- Wähle immer neue Zahl  $k$ ! Wenn man gleiches  $k$  für zwei verschiedene Nachrichten  $m_1, m_2$  nimmt, kann man  $k$  und damit auch  $x$  bestimmen:  $s_1 - s_2 \equiv (m_1 - m_2)k^{-1} \pmod{p-1}$  und  $k = (s_1 - s_2)^{-1}(m_1 - m_2) \pmod{p-1}$
- Man muß eine Hashfunktion verwenden, da man sonst ElGamal Unterschriften fälschen kann. Man kann allerdings nicht  $m$  frei wählen.
  1. Wähle  $b, c$  mit  $\text{ggT}(c, p-1) = 1$
  2.  $r := g^b y^c \pmod{p}$
  3.  $s := -rc^{-1} \pmod{p-1}$
  4.  $m := -rbc^{-1} \pmod{p-1}$

Die Unterschrift erfüllt dann:  $g^m \equiv y^r r^s \pmod{p}$ , da  $g^{-rbc^{-1}} \equiv y^{s^b y^c} (g^b y^c)^{-rc^{-1}} \pmod{p}$   
 und  $g^{-rbc^{-1}} \equiv y^{s^b y^c} g^{-rbc^{-1}} y^{-(g^b y^c)cc^{-1}} \pmod{p}$

### 3.6 Elliptische Kurven

#### Elliptische Kurven

- $f(x) = a \cdot x + c$
- $y = a \cdot x + c$
- $a \cdot x - y + c = 0$
- $a \cdot x - b \cdot y + c = 0$
- $a \cdot x^2 + b \cdot y^3 + c = 0$
- $a_1 \cdot x^2 a_2 \cdot x * 3 + b_1 \cdot y^3 b_2 \cdot y + c = 0$
- Gleichungen über  $\mathbb{R}$  oder  $\mathbb{Q}, \mathbb{Z}, \mathbb{F}, \dots$

#### Notation:

$f(x, y)$  ein Polynom über zwei Variablen

$c \cdot x^m \cdot y^n$  ein Term von  $f(x, y)$

$c \neq 0$  konstant

**Grad** eines Terms gegeben durch  $n + m$

**Grad von**  $f(x, y)$  ist größter Grad der Terme

**Beobachtungen:**  $f(x, y)$  Polynom vom Grad  $d$  und  $g(x, y)$  Gerade ( $a \cdot x + b \cdot y + c = 0$ ), dann schneidet die Gerade den Graph von  $f(x, y) = 0$  höchstens in  $d$  Punkten. Beweis: Löse Geradengleichung nach  $x$  auf und setze  $f(x, y)$  ein, dann gibt es für  $y$  max.  $d$  Nullstellen.

## AUFBAU EINER GRUPPE

Ziel: Gruppenoperationen auf Graphen von  $f(x, y) = 0$

- Operation: Zu zwei Punkten auf  $f(x, y) = 0$  bestimme dritten Punkt auf  $f(x, y) = 0$
- Ansatz: Nutze Gerade zur Bestimmung des dritten Punktes, die Gerade durch zwei Punkte  $P, Q$  bestimmt den dritten Punkt. Wähle  $f(x, y) = 0$  vom Grad 3
- Gerade:  $g(x, y) = a \cdot x + b \cdot y + c = 0 : b \neq 0$
- Lösung:  $f(x, (-a \cdot x - c)/b) = 0$  ist kubische Gleichung in  $x$  und hat 3 Lösungen, von denen zwei bekannt sind und die Dritte den neuen Punkt ( $x$ -Koordinate) ergibt. Über die Geradengleichung ergeben sich die Punkte.
- Operation  $P \oplus Q = R$

**Weierstrass Form:** Eine elliptische Kurve ist der Graph  $E$  bzw.  $E_{a,b}$  der Gleichung  $y^2 = x^3 + a \cdot x + b : x, y, a, b \in \mathbb{R}(\mathbb{Q}, \mathbb{Z}, \mathbb{N}_m)$ . Weiterhin gehört zu  $E$  der Punkt  $\infty$ . Dabei wird  $\infty$  das neutrale Element.

## GRUPPENDEFINITION

Sei  $P = (x, y)$  ein Punkt von  $y^2 = x^3 + a \cdot x + b$

- $-P = (x, -y)$
- Falls  $P = (x, y)$  und  $Q = (x, -y) = -P$  dann setze  $P + Q = \infty$
- Andernfalls ist  $P + Q = -R : R = P \oplus Q$
- $P + \infty = \infty + P = P$
- $\infty + \infty = \infty + \infty = \infty$
- $P + P$  wie folgt def.:
  - Bestimme Tangente  $T$  im Punkt  $P$  an  $E$
  - Falls  $T$  vertikal ist:  $P + P = \infty$
  - Sei  $R$  weiterer Schnittpunkt von  $T$ , setze  $P + P = -R$

Aussagen:

- Die oben def. Operation auf  $y^2 = x^3 + a \cdot x + b$  ist eine Gruppe
- Die Gruppe auf  $y^2 = x^3 + a \cdot x + b \pmod{p}$  hat  $N = p + 1 \sum_{r=0}^{p-1} \frac{x^3 + a \cdot x + b}{p}$  Elemente
- (Hasse Theorem): Die Gruppe hat  $N$  Elemente mit  $p + 1 - 2 \cdot \sqrt{p} < N < p + 1 + 2 \cdot \sqrt{p}$
- Und der disk. Log. auf dieser Gruppe ist schwer.

**Darstellung der Blöcke:**

**probalistisches Verfahren**

- $E, p, N$  wie oben,  $M$  Nachrichtenblock mit  $0 < M < p/s^k$
- $x = 2^k \cdot M + i$   $x$ -Koordinate für ein  $i : 0 \leq i < 2^k$ , so daß es einen Punkt  $(x, y) \in E$  gibt.
  - $\forall i : 0 \leq i < 2^k$
  - $x = 2^k \cdot M + i$
  - Falls  $(\frac{x^3 + a \cdot x + b}{p}) = +1$  gebe  $i$  aus
  - Gebe Fehler aus
- Fehlerwahrscheinlichkeit ist:  $1/2^{2^k}$

## deterministisches Verfahren

- Wähle  $p \equiv 3 \pmod{4}$
- $-1$  ist der quadratische Nichtrest, d.h.  $\left(\frac{-1}{p}\right) = -1$
- Setze  $b = 0$
- Dann ist  $E$  geg. durch  $y^2 = x^3 + a \cdot x \pmod{p}$
- $M$  zu verschlüsselnde Nachricht mit  $0 < M < p/2$  und wir verlieren ein Bit.
- Bestimme  $t = M^3 + aM \pmod{p} \Rightarrow t \vee -t$  ist quadratischer Rest
- Falls  $\left(\frac{t}{p}\right) = 1$ , dann setze  $x = M$
- Falls  $\left(\frac{-t}{p}\right) = 1$ , dann setze  $x = p - M$
- Damit gilt  $\left(\frac{x^3 + a \cdot x}{p}\right) = 1$  und  $y$  kann bestimmt werden
- Entschlüsseln:  $M = \min(x, p - x)$

## Verschlüsseln mit elliptischen Kurven:

### Idee

- Gehe Analog zu ElGamal vor
- Problem: Wie werden die Nachrichten auf Elemente der Gruppe abgebildet? Dazu 2 Verfahren:
  - probabilistisches Verfahren, welches  $k$  Bits *verschwendet*
  - deterministisches Verfahren, welches 1 Bit *verschwendet*, aber eine spezielle Gruppe braucht.

### Aufbau

- Allgemein (alles öffentlich)
  - Bestimme Primzahl  $p$
  - Bestimme elliptische Kurve  $E \pmod{p}$
  - Bestimme einen beliebigen Punkt  $P_0$  auf  $E$  mit großer Ordnung
- Nutzer  $A$ 
  - Bestimme Zahl  $a_A$  (sein privater Schlüssel)
  - Bestimme  $P_A = a_A P_0$
  - Veröffentliche pub. key  $P_A$

## Ver- und Entschlüsselung

- $B$  sendet an  $A$ 
  - Bestimme Punkt  $P$  aus Nachricht  $M$  mit einem der zwei Verfahren
  - Wähle zufällig eine Zahl  $k : 0 < k < p$
  - Sende an  $A$ :  $(kP_0, kP_A + P)$  wie bei ElGamal (Hilfe, verschlüsselter Text)
- Entschlüsselung:
  - $kP_A + P - (ka_A)P_0 = (ka_A)P_0 + P - (ka_A)P_0 = P$
- Falls diskreter Logarithmus auf  $E$  lösbar:
  - bestimme  $k$  aus  $kP_0$
  - bestimme  $kP_A$
  - bestimme  $kP_A + P - kP_A$

## ÜBERBLICK VERFAHREN: ELLIPTISCHE KURVEN

**A**  $a_A, P_A, P_0, E \rightarrow$  **B** Wähle Primzahl  $p$ , ell. Kurve  $E(\text{mod } p)$ ,  $P_0 \in E$ ,  $a_A \in \{1, \dots, p-1\}$ . Bestimme  $P_A := a_A P_0$ . Und sende  $P_0, E, P_A$ .

**B**  $m, P_0, E, P_A \rightarrow$  **A** Erzeuge  $P \in E$  aus  $m$ . Wähle  $k \in \{1, \dots, p-1\}$ . Übertrage  $T_1 = kP_0$  und  $T_2 = kP_A + P$

**A** Berechne  $T' = T_2 - a_A T_1 = kP_A + P - (a_A k)P_0 = (a_A k)P_0 + P - (a_A k)P_0$  und bestimme  $m$  aus  $T'$

### Zusammenfassung:

- Analoges Ansatz wie ElGamal
- kompliziertere Gruppe als Basis
- Mindestens analoge Sicherheit

## 3.7 Quantenkryptographie

- Öffentliche Übertragung von Bitvektoren. Nachher kann man mit sehr hoher Wahrscheinlichkeit sagen, ob jemand gelauscht hat
- Keine Verschlüsselung im eigentlichen Sinne, eher ein Schlüsselaustausch für One-Time-Pad

### Ablauf:

- A und B wählen jeweils unabhängig voneinander eine Reihe von Filtereinstellungen (2 Möglichkeiten)
- A sendet daraufhin eine Anzahl von Photonen an B. B empfängt ein Photon, wenn die aktuelle Filtereinstellung passt
- Wird ein Photon von B empfangen, zählt er dies als 1, wenn nicht, als 0
- Sind alle übertragen, tauschen beide ihre Filtereinstellungen aus und prüfen, was A geschickt hat bzw was B empfangen hat. Beide kennen also die Bitfolge des anderen
- Ein potentieller Lauscher hat erstmal die gleichen Eigenschaften von B. Aber da er wie B auch seine Filter wählt, verursacht er den gleichen Fehler wie B
- Sitzt nun ein Lauscher C zwischen A und B, so ist die Wahrscheinlichkeit, dass B ein Photon korrekt empfängt nicht mehr  $1/2$  sondern  $1/4$ . Und das können A und B verifizieren
- Zur Verifikation sendet B eine Auswahl (etwa die Hälfte) der korrekt erkannten Bits an A
- Mit den restlichen (also ca.  $1/4$  der insgesamt gesendeten Bits) wird nun per One-Time-Pad verschlüsselt

## 3.8 Hilfsfunktionen

HIER FEHLT WAS ZU OAEP, HASHFUNKTIONEN (MERKLES-META) UND PRIMZAHLTESTS (SOLOWAY-STRASSEN, MILLER-RABIN, DETERMINISTISCHER TEST)

## 4 Protokolle

### 4.1 Quittungen

#### Quittung 1

- A sendet  $E_B(w)$  zusammen mit 'A' und 'B' an B ("Ich bin A und sende dir, B, einen Text")
- B entschlüsselt  $w$  und sendet wieder 'A', 'B' und  $E_A(w)$  an A
- A entschlüsselt  $w$  wieder und überprüft, ob das  $w$  korrekt ist

Ein aktiver Lauscher kann hier das  $w$  herausbekommen

## Quittung 2

Als Verbesserung wird nun der Empfänger mit verschlüsselt

- A sendet 'A' als Ansender und  $E_B(E_B(w), B)$  an B
- B entschlüsselt zwei mal und sendet  $E_A(E_A(w), A)$  an A
- A entschlüsselt zwei mal und überprüft  $w$

Ein Lauscher kann hier ebenfalls an das  $w$  gelangen wenn A und B sich nicht an ein paar Regeln halten:

- Nachrichten speichern
- Zwei verschiedene Verschlüsselungen wählen:  $E_B(E_{B'}(w), B)$
- Nur sinnvolle Texte zulassen
- ...

NICHT VOLLSTÄNDIG

## 4.2 Elektronische Wahlen I

### Einleitung

**Anforderungen** anonym, geheim, freie Entscheidung ob man wählt, diese Tatsache auch geheim, Wahlen nachvollziehbar, keine Mehrfachwahl möglich, Erkennung falscher Stimmen, Wiedererkennung der eigenen Stimme, eigene Stimme nicht zeigbar, keine hohen Kosten

**Möglichkeiten** vertrauenswürdige Person zum Testen der Identität und zur Auszählung (Gefahr der Zuordnung Stimme  $\leftrightarrow$  Wähler), zwei vertrauenswürdige Personen (Ausähler C und Legitimator L), Auszählung wird von Teilnehmern übernommen, Auszählung wird von überwiegend vertrauenswürdigen Personen ausgeführt

### Protokoll mit $L, C$ und $n$ Wählern $A_j$

- Wähler  $A_j$  identifiziert sich bei  $L$ .  $L$  erzeugt einen Wahlzettel  $i(A_j)$ , der für  $C$  nicht zu  $A_j$  zurückverfolgbar ist und leitet das an  $A_j$  und  $C$  weiter.
- $A_j$  wählt seinen Kandidaten (mit seinem Wahrschein)  $v(A_j)$ , bestimmt eine geheime Zufallszahl  $s(A_j)$  und verschlüsselt beides zusammen mit  $i(A_j)$  als Vektor zuerst für  $C$  und danach für  $L$ .
- $L$  entschlüsselt das nun. Am Ende schickt  $L$  die Menge der  $i(A_j)$  und die Menge der entschlüsselten Stimmen an  $C$ .
- $C$  kann nun nochmal entschlüsseln und vergleichen, ob die jeweilige Stimme gültig war und für wen gestimmt wurde. Sofern  $L$  und  $C$  nicht miteinander arbeiten, kann kein Zusammenhang zwischen Wähler und Stimme gefunden werden.
- $C$  veröffentlicht für jeden Kandidaten eine Liste der  $s(A_i)$ , womit jeder Wähler kontrollieren kann, ob seine Stimme richtig gezählt wurde.

Es wurden aber noch nicht alle oben genannten Punkte erfüllt.

### Protokoll mit $L$ und $n$ Wählern $A_j$

- $A_j$  identifiziert sich bei  $L$
- $L$  erstellt eine endliche Menge  $S$  und eine Einwegpermutation  $f$  (ist eigentlich eine Einwegfunktion die injektiv ist) auf  $S$  sowie  $n^k$  Tokens.  $A_j$  kauft nun eines der Tokens geheim. O.B.d.A. sei dies  $t_i$ .
- $A_j$  bestimmt nun ein zufälliges  $s \in S$  und wendet darauf die Funktion  $f$  an. Desweiteren wählt er einen Kandidaten  $v_j$
- Anonym schickt  $A_j$  nun sein Token, seine Wahl und  $f(s)$  an  $L$ .  $L$  testet nun, ob das Token korrekt ist. Wenn ja, dann veröffentlicht er die Wahl und  $f(s)$ .
- Soll ein anderer Kandidat gewählt werden, so kann einfach ein anderes  $s'$  gewählt werden, auf das wieder die Funktion  $f$  angewendet wird. Dieses  $f(s')$  wird zusammen mit der neuen Wahl und dem alten Token an  $L$  geschickt.
- Ermöglicht Online-Wahlen, also man kann jederzeit wählen, nicht nur alle 4 Jahre oder so.

### Protokoll ohne Zentrale, Auswertung durch Wähler Nur für kleine Wählergruppen, Berechnungen werden schnell sehr groß

Idee: Eigene Stimme verschlüsseln, Stimmen mischen, weitergeben

Hier: jede Partei  $X \in \{A, B, C, D\}$  hat einen öffentlichen und einen privaten Schlüssel

- Jede Partei erzeugt folgendes:  $E_A(R_5 E_B(R_4 E_C(R_3 E_D(R_2 E_A(E_B(E_C(E_D(v_x R_1))))))))$  mit  $v_x$  als Wahl von User  $X$  und Zufallszahlen  $R_i$
- Jedes Zwischenergebnis wird gespeichert. A sammelt nun die Nachrichten, entschlüsselt diese, entfernt die Zufallszahl und permutiert sie daraufhin.
- Die Nachrichten werden nun unterschrieben an B geschickt, der genauso verfährt. Das Ganze läuft zweimal durch die Wählergruppe. Der letzte gibt das Ergebnis bekannt

### Protokoll mit $n$ Wählern $A_j$ , $k$ Mischern $S_j$ , einem Zentrum $C$ und einem öffentlichem Medium $B$

- $C$  bestimmt eine große Primzahl  $p$ , einen Generator  $g$  in  $\mathbb{Z}_*^q$  mit  $q = p^k$  und eine Funktion  $\phi: \mathbb{Z}_q^* \rightarrow \mathbb{K}$  und veröffentlicht das alles unterschrieben auf  $B$
- Jeder Mischer  $S_i$  wählt nun ein  $a_i$  mit  $ggT(a_i, q - 1) = 1$  und berechnet damit  $c_i = g^{a_i}$ . Das veröffentlicht er unterschrieben auf  $B$
- Nun nimmt der erste Mischer  $S_1$  den Generator  $g$ , setzt ihn gleich  $z_0$  und berechnet  $z_1 = z_0^{a_1}$ .  $z_0$  und  $z_1$  veröffentlicht er unterschrieben auf  $B$ . Der nächste Mischer  $S_2$  holt sich dieses Tupel, nimmt sich  $z_1$ , berechnet daraus  $z_2 = z_1^{a_2}$  und veröffentlicht  $z_1$  und  $z_2$  unterschrieben auf  $B$ . Das Ganze läuft nun so weiter bis zum letzten Mischer  $S_k$ , der  $z_{k-1}$  und  $z_k$  veröffentlicht.
- Nun beweist jeder Mischer, dass er richtig gerechnet hat, indem er beweist, dass zwei diskrete Logarithmen gleich sind:  $c_i = g^{a_i}$  und  $z_i = z_{i-1}^{a_i}$
- Jeder Wähler wählt nun einen Kandidaten  $K_i$ , indem er solange ein  $b_i$  rät, so dass die folgende Gleichung erfüllt ist:  $\phi(z_k^{b_i}) = \phi(g^{a_1 a_2 a_3 \dots a_k b_i}) = K_i$ . Unterschrieben veröffentlicht er nun  $v_i = g^{b_i}$
- $C$  löscht nun am Ende der Wahl alle ungültigen Stimmen. Danach holt er sich alle Stimmen  $v_i$ , permutiert sie und schickt sie wieder an  $B$
- Die Mischer holen sich nun nacheinander die Menge der Stimmen, potenzieren jede Stimme mit ihrem Geheimnis  $a_i$ , permutieren die Menge und schicken diese wieder an  $B$
- Zur Auswertung holt sich  $C$  nun die  $k$  mal permutierten und mit  $a_1 a_2 a_3 \dots a_k$  potenzierten Stimmen und prüft dann für jeden Kandidaten  $K_i$ , ob  $\phi(\text{jede einzelne Stimme}) = K_i$  gilt

Sicherheitsaspekte:

- Stimmen werden richtig gezählt
- Wähler können das Mischen überwachen
- Solange es mindestens einen ehrlichen Mischer gibt, bleiben die Stimmen geheim
- Aus den gemischten Werten kann nicht auf die Stimmen geschlossen werden
- Ändern der Stimme ist möglich, aber aufwendig
- Aber: Wähler kann einem Dritten seine Wahl preisgeben

#### Protokoll wie oben, nur ohne Stimmenkauf

- C baut das System ähnlich wie oben auf, nur existiert nun keine  $\phi$ -Funktion und  $q$  ist eine Primzahl. Der Generator  $g$  und  $q$  werden wieder unterschrieben auf  $B$  veröffentlicht
- Die Mischer  $S_i$  wählen wieder ein  $a_i$  mit den o.g. Eigenschaften, berechnen daraus wieder ihr  $c_i$  und veröffentlichen dies wieder unterschrieben
- C bestimmt nun  $t$  Tokens ( $t$  ist viel größer als  $n$  = Anzahl der Wähler) und veröffentlicht diese auf  $B$
- Jeder Mischer bestimmt nun eine Permutation, und nacheinander wendet jeder seine Permutation auf die Menge der Tokens an und potenziert jedes Element mit seiner Zahl  $a_i$ . Die Mischer beweisen ihre Rechenarbeit durch einen ZKP
- Am Ende sind dann alle Tokens vermischt und verschlüsselt. Aus diesen wählt C nun eines und sendet es an einen Wähler  $A_j$ . Gleichzeitig bestimmen die Mischer eine gemeinsame Verschlüsselung  $E$ , die auf  $B$  veröffentlicht wird.
- Der Wähler  $A_j$  schickt nun sein Token nacheinander an jeden Mischer, der dieses Token mit seinem  $a_i$  potenziert (zB wird an  $S_6$  das geschickt, was  $A_j$  von  $S_5$  bekommen hat). Als Verbesserung kann  $A_j$  das Token jeweils erst mit  $g^{r = \text{Zufallszahl}}$  multiplizieren und nachdem er das Token wieder zurück bekommen hat, mit  $c_{\text{Index des Mischers}}^{-r}$  multiplizieren
- Der Wähler  $A_j$  verschlüsselt nun das Tupel ( $v_j = \text{Stimme, Token vom letzten Punkt}$ ) mit  $E$  und schickt es anonym an  $B$
- Die Mischer holen sich jetzt die Tupel, entschlüsseln sie und senden sie wieder an  $B$
- Nun holt sich C die entschlüsselten Tupel, sortiert die Tokens in Kandidatenlisten (für jeden Kandidaten eine Liste an Tokens), permutiert diese und veröffentlicht diese Mengen wieder auf  $B$
- Die Mischer nehmen sich nun nacheinander die Menge der Tokens von  $B$ , machen zweimal die Schritte (Permutieren der Menge und Potenzieren mit dem eigenen  $a_i$ ) und veröffentlichen die Menge daraufhin wieder auf  $B$  (mit Beweis der Korrektheit der Rechnung).
- In der letzten Menge sind nun die ursprünglichen Token vorhanden. Die Zählung ist nun einfach und öffentlich möglich.

### 4.3 Elektronische Wahlen II

#### $(t, n)$ -Threshold-Scheme und Homomorphe Verschlüsselung

- Wahlsystem, bei dem verschlüsselte Stimmen addiert werden
- Wahlsystem, mit Ausfallsicherheit.

**Definition** ( $(t, n)$ -Threshold-Scheme ( $t \leq n$ )). Ist ein System aus  $n$  Teilnehmern mit folgenden Eigenschaften:

1. Jeder Teilnehmer  $i \in T$  hat ein Geheimnis  $s_i$
2. Es gibt ein daraus zu bestimmendes Geheimnis  $s$

3. Jede Teilmenge  $T$  von Teilnehmern mit  $|T| \geq t$  kann das Geheimnis  $s$  gemeinsam bestimmen
4. Jede Teilmenge  $T$  von Teilnehmern mit  $|T| < t$  kann das Geheimnis  $s$  *nicht* gemeinsam bestimmen.

Ein solches System kann man über Polynome vom Grad  $t - 1$  aufbauen.

**Lemma 2.** Sei  $f(X) = \sum_{i=0}^{t-1} a_i X^i \in \mathbb{Z}_p[X]$  ein Polynom vom Grad  $t - 1$ . Weiter sei:

$$\mathcal{P} := \{(x_i, f(x_i)) \mid x_i \in \mathbb{Z}_p, i = 1, \dots, t, x_i \neq x_j \text{ für } i \neq j\}$$

Für  $\mathcal{Q} \subset \mathcal{P}$  setze:  $\mathcal{P}_{\mathcal{Q}} := \{g \in \mathbb{Z}_p[X] \mid \deg(g) = t - 1, g(x) = y, \forall (x, y) \in \mathcal{Q}\}$ . Dann gilt:

1.  $\mathcal{P}_{\mathcal{P}} = \{f(X)\}$
2. Falls  $\mathcal{Q} \subsetneq \mathcal{P}$  und  $x \neq 0 \forall (x, y) \in \mathcal{Q}$ , dann ist die Verteilung der konstanten Anteile der Polynome gleich der eines  $a \in \mathbb{Z}_p$ .  
D.h. die konstanten Anteile der Polynome sind nicht zu unterscheiden.

**Lemma 3.** Sei  $f(X)$  ein Polynom vom Grad  $t - 1$  und weiter  $\mathcal{P} := \{(x_i, f(x_i)) \mid i = 1, \dots, t, x_i \neq x_j \text{ für } i \neq j\}$ . Dann gilt (Lagrange Interpolation):  $f(X) = \sum_{i=1}^t f(x_i) \prod_{1 \leq j \leq t, i \neq j} \frac{X - x_j}{x_i - x_j}$

*Beweis.*

1. Rechte Seite ist ein Polynom vom Grad  $t - 1$
2. Falls man dort  $X$  durch  $x_i$  ersetzt, dann gilt  $f(x_i) = g(x_i)$
3. Wegen der Eindeutigkeit des Polynoms folgt dann die Behauptung □

### SHAMIRS $(t, n)$ -THRESHOLD-SCHEME AUFBAU

Ein vertrauenswürdigen Zentrum  $T$  führt folgende Schritte durch:

1. Wählt Geheimnis  $s$
2. Wählt  $p$  Primzahl mit  $p > \max(s, m)$  und setzt  $a_0 = s$
3. Wählt  $a_1, a_2, \dots, a_{t-1} \in \{0, \dots, p - 1\}$  zufällig
4. Setze  $f(X) = \sum_{i=0}^{t-1} a_i X^i$
5. Bestimme  $s_i = f(i), i \in \{1, \dots, n\}$
6. Sendet  $(i, s_i)$  an Teilnehmer  $P_i$

Sei  $J \subset \{1, \dots, n\}$  mit  $|J| \geq t$ , dann gilt:  $s = a_0 = f(0) = \sum_{i \in J} f(i) \prod_{j \in J, j \neq i} \frac{j}{j-i} = \sum_{i \in J} s_i \prod_{j \in J, j \neq i} \frac{j}{j-i}$

*Anmerkung.*

1. Das Shamir  $(t, n)$ -Threshold-Scheme ist perfekt: mit weniger als  $t$  Teilnehmer kann keine Information über  $s$  bestimmt werden.
2. Es ist leicht um neue Teilnehmer erweiterbar, d.h.  $n$  kann leicht erhöht werden.
3. Zur Gewichtung kann ein Teilnehmer mehrere Geheimnisse bekommen.
4. Es ist schwer,  $t$  zu ändern, da dafür das ganze System neu aufgebaut werden muß.

### Homomorphe Verschlüsselung

**Erinnerung:** Aufbau:  $p, q$  Primzahlen,  $q - 1$  teilt  $p - 1$ ,  $G$  Untergruppe der Ordnung  $q - 1$  in  $\mathbb{Z}_p^*$ ,  $g, v$  Generatoren in  $G$  zufällig.

$\mathbf{P} \rightarrow \mathbf{V}$   $m \in \{0, \dots, q - 1\}$  wähle  $r \in \{0, \dots, q - 1\}$  und sende  $c := g^r v^m \pmod p$  nach  $\mathbf{V}$

$\mathbf{P} \rightarrow \mathbf{V}$  Anschließend übertrage  $r, m$

$\mathbf{V}$  Teste  $c \stackrel{?}{\equiv} g^r v^m \pmod p$

Wir bezeichnen nun  $Com(r, m) := g^r v^m$

**Homomorphe Systeme:** Sei  $r_1, r_2, m_1, m_2 \in \{0, \dots, q-1\}$ , dann gilt:

$$Com(r_1, m_1) \cdot Com(r_2, m_2) = g^{r_1} v^{m_1} g^{r_2} v^{m_2} = g^{r_1} g^{r_2} v^{m_1} v^{m_2} = g^{r_1+r_2} v^{m_1+m_2} = Com(r_1+r_2, m_1+m_2)$$

- Damit haben wir ein homomorphes Commitment-Schema welches wir als Grundlage für Wahlen verwenden können
- Stimme wird in  $m_i$  kodiert und das Auszählen geht ohne das Öffnen der Einzelstimmen.

**Homomorphe Verschlüsselung:**

- Wähler  $i$  wählt  $m_i \in \{0, 1\}$ , Zufallszahl  $r_i \in \{0, \dots, q-1\}$ , bestimmt  $c_i := g^{r_i} v^{m_i}$  und veröffentlicht  $c_i$
- Wähler  $i$  verschlüsselt für  $T$  den Wert  $E_T(g^{r_i})$
- $T$  bestimmt:  $D_T(\prod_{i=1}^n E_T(g^{r_i})) = \prod_{i=1}^n g^{r_i} = g^{\sum_{i=1}^n r_i}$
- $T$  veröffentlicht:  $g^{\sum_{i=1}^n r_i}$
- Jeder Wähler kann bestimmen:  $\frac{\prod_{i=1}^n c_i}{g^{\sum_{i=1}^n r_i}} = \frac{\prod_{i=1}^n g^{r_i} v^{m_i}}{g^{\sum_{i=1}^n r_i}} = \frac{g^{\sum_{i=1}^n r_i} v^{\sum_{i=1}^n m_i}}{g^{\sum_{i=1}^n r_i}} = v^{\sum_{i=1}^n m_i}$

**Wahlssysteme mit Zentrum**

**Einleitung:** Wir betrachten nur **ja/nein** Wahlen. Das System wird durch ein vertrauenswürdiges **Zentrum** aufgebaut, welches die  $n$  Auszähler bestimmt. Das System ist **sicher** solange sich nicht  $t$  der Auszähler zusammenschließen. Außerdem ist es **robust**: Solange  $t$  der Auszähler korrekt arbeiten und keiner absichtlich falsche Informationen abliefern, kann die Wahl durchgeführt werden. Die Kommunikation erfolgt **öffentlich** über ein Bulletin Board. Beteiligt sind das Zentrum  $T$ ,  $n$  Auszähler  $A_1, \dots, A_n$  und  $m$  Wähler  $V_1, \dots, V_m$

**Aufbau: System:**

$\mathbf{T} \xrightarrow{p, h, g} \mathbf{B}$   $T$  wählt  $p, q$  Primzahlen,  $q$  großer Teiler von  $p-1$ ,  $G$  Untergruppe von  $\mathbb{Z}_p^*$  der Ordnung  $q-1$ . Wählt  $g$  Generator in  $G$  und  $s \in \{0, \dots, q-1\}$  zufällig. Und schreibt dann  $p, h := g^s, g$  aufs Board.

Also haben wir ein ElGamal System.

**Aufbau: Eigenschaften:**

- Ein  $m \in G$  wird verschlüsselt mit  $(c_1, c_2) = (g^\alpha, h^\alpha m)$  ( $\alpha \in \{0, \dots, q-1\}$  zufällig gewählt)
- Die Entschlüsselung erfolgt über  $s$  durch  $m = c_2 c_1^{-s}$
- Die Verschlüsselung ist homomorph. (läßt sich leicht nachrechnen  $(c_1, c_2)(c'_1, c'_2) = \dots = (g^{\alpha+\alpha'}, h^{\alpha+\alpha'} m m')$ )

**Aufbau: Aufbau:**

$\mathbf{T} \xrightarrow{(j, h_j)} \mathbf{B}$   $T$  bestimmt Shamir  $(t, n)$ -Threshold-Scheme, also  $(j, s_j)$  für  $1 \leq j \leq n$ ,  $h_j := g^{s_j}$  für  $1 \leq j \leq n$  und schreibt  $(j, h_j) | 1 \leq j \leq n$  aufs Board.

Dann werden noch die Teilgeheimnisse über sicheren Kanal an die Auszähler verteilt:

$\mathbf{T} (j, s_j, h_j) \xrightarrow{(j, s_j)} A_j$   $T$  überträgt  $(j, s_j)$  an  $A_j$

**Aufbau: Entschlüsselung:** Damit kann nun ein  $(c_1, c_2) = (g^\alpha, h^\alpha m)$  mit Hilfe der  $A_j$  entschlüsselt werden, ohne  $s$  explizit zu bestimmen

$A_j (j, s_j) \xrightarrow{(c_1, c_2)} \mathbf{B} (c_1, c_2)$   $A_j$  liest  $(c_1, c_2)$  vom Board.

$A_j \xrightarrow{w_j} \mathbf{B}$   $A_j$  schreibt  $w_j := c_1^{s_j}$  aufs Board.

$\mathbf{B} J = \{j | A_j \text{ ehrlich geantwortet}\}, c_1^s = \prod_{j \in J} w_j^{\prod_{l \in J, l \neq j} \frac{1}{l-j}}, m = c_2 c_1^{-s}$

**Aufbau: Überblick:** Für das Wahlsystem benutzen wir als Idee das vorgestellte System zur Entschlüsselung: Jeder Wähler  $V_i$  wählt  $\alpha_i$  zufällig und wählt  $v_i \in \{-1, 1\}$  und verschlüsselt anschließend seine Wahl durch  $g^{v_i}$  (d.h. durch  $c_i = (c_{i,1}, c_{i,2}) = (g^{\alpha_i}, h^{\alpha_i} g^{v_i})$ ). Dann veröffentlicht er dies mit einer Unterschrift und es wird weiterhin abgesichert, dass  $v_i \in \{-1, 1\}$  gilt.

**Aufbau: Protokoll:**

$V_i \xrightarrow{c_i} \mathbf{B}$   $V_i$  wählt  $v_i \in \{-1, 1\}$  und schreibt  $c_i = (c_{i,1}, c_{i,2}) = (g^{\alpha_i}, h^{\alpha_i} g^{v_i})$  aufs Board.

$V_i \rightarrow \mathbf{B}$   $V_i$  unterschreibt  $c_i$  auf dem Board.

$V_i \rightarrow \mathbf{B}$   $V_i$  beweist  $v_i \in \{-1, 1\}$  auf dem Board.

**Aufbau: Auszählen:**

$A_j (j, s_j) \xleftarrow{(c_1, c_2)} \mathbf{B}$   $c_i$   $\mathbf{B}$  kennt die  $c_i$  wobei  $c$  die Verschlüsselung von  $g^d | d = \sum_{i=1}^m v_i$ .  $A_j$  liest  $(c_1, c_2)$  vom Board.

$A_j \xrightarrow{w_j} \mathbf{B}$   $A_j$  schreibt  $w_j := c_1^{s_j}$  aufs Board.

$A_j$  beweist  $w_j := c_1^{s_j} | J = \{j | A_j \text{ ehrlich geantwortet}\}$ ,  $c_1^s = \prod_{j \in J} w_j^{\prod_{l \in J, l \neq j} \frac{1}{1-l}}$ ,  $g^d = c_2 c_1^{-s}$ .  $\forall x \in \{-m, \dots, m\}$  teste  $g^x \stackrel{?}{=} c_2 c_1^{-s}$ . Wahlergebnis  $e$  mit:  $g^e \stackrel{?}{=} c_2 c_1^{-s}$

Anmerkung.

1. Der Test über die Ehrlichkeit von  $T_i$  und  $V_i$  wird im Folgenden betrachtet.
2. Die Sicherheit und Ausfallsicherheit ergibt sich über Shamirs  $(t, n)$ -Threshold-Scheme
3. Weitere Sicherheit ergibt sich über das verwendete ElGamal Verschlüsselungsverfahren (Schwierigkeit des Diffie-Hellman-Problems)

**Test der Angaben der Auszähler  $A_j$ :**

1. Die  $A_j$  müssen zeigen, dass  $w_j \stackrel{?}{=} c_1^{s_j}$  gilt, wobei bekannt ist, dass  $h_j = g^{s_j}$
2. Also gilt es zu zeigen, dass der diskrete Logarithmus für  $h_j$  und  $w_j$  gleich ist, also ist ein Protokoll anzugeben, das zeigt, dass  $y_1 = g_1^x$  und  $y_2 = g_2^x$  gilt.

**Protokoll:**

$\mathbf{P} x, y_1, y_2, g_1, g_2, q \rightarrow \mathbf{V} y_1, y_2, g_1, g_2, p, q$  Es gilt  $y_1 = g_1^x$ ,  $y_2 = g_2^x$ .  $r \in \{0, \dots, q-1\}$  zufällig.  $\mathbf{P}$  überträgt  $a := (a_1, a_2) := (g_1^r, g_2^r)$  an  $\mathbf{V}$ .

$\mathbf{P} \leftarrow \mathbf{V}$   $\mathbf{V}$  überträgt zufälliges  $c \in \{0, \dots, q-1\}$  an  $\mathbf{P}$ .

$\mathbf{P} \rightarrow \mathbf{V}$   $\mathbf{P}$  überträgt  $b := r - cx$  an  $\mathbf{V}$ .

$\mathbf{V}$  Testet dann ob gilt:  $a_1 \stackrel{?}{=} g_1^b y_1^c \pmod{q}$  und  $a_2 \stackrel{?}{=} g_2^b y_2^c \pmod{q}$

**Lemma.** Wenn  $\mathbf{P}$   $x$  kennt, dann kann  $\mathbf{P}$   $\mathbf{V}$  überzeugen und wenn  $\mathbf{V}$  mit Wahrscheinlichkeit  $> 1/p$  überzeugt wird, dann kennt  $\mathbf{P}$   $x$ .

**Betrugsversuch:**  $\mathbf{P}'$  versucht zu betrügen, indem zwei verschiedene Zufallszahlen benutzt werden.

$\mathbf{P} \rightarrow \mathbf{V}$   $\mathbf{P}$  wählt  $r, \tilde{c} \in \{0, \dots, q-1\}$  zufällig und schickt  $a := (a_1, a_2) := (g_1^r y_1^{\tilde{c}}, g_2^r y_2^{\tilde{c}})$  an  $\mathbf{V}$ .

$\mathbf{P} \leftarrow \mathbf{V}$   $\mathbf{V}$  challenged das dann mit einem zufälligen  $c \in \{0, \dots, q-1\}$ .

$\mathbf{P} \rightarrow \mathbf{V}$   $\mathbf{P}$  antwortet dann normal mit  $b := r - cx$ .

$\mathbf{V}$  testet dann wie gehabt:  $g_1^r y_1^{\tilde{c}} \equiv a_1 \stackrel{?}{=} g_1^b y_1^c \pmod{q}$  und  $g_2^r y_2^{\tilde{c}} \equiv a_2 \stackrel{?}{=} g_2^b y_2^c \pmod{q}$ . Betrug ist erfolgreich, falls  $c = \tilde{c}$

Könnte  $\mathbf{P}'$  mit Wahrscheinlichkeit  $> 1/p$  betrügen, dann könnte er den diskreten Logarithmus berechnen, oder er betrügt sich selbst.

*Anmerkung.* Es ist nicht bekannt, ob das Protokoll ZKP ist, aber es ist ein *Ehrlicher-Verifizierer ZKP*. D.h. es ist ZKP wenn man annimmt, dass  $V$  immer  $c \in \{0, \dots, q-1\}$  wirklich zufällig wählt. Dann ist folgende Simulation einer Kommunikation möglich:

1. Wähle  $\tilde{b} \in \{0, \dots, q-1\}$  zufällig
2. Wähle  $\tilde{c} \in \{0, \dots, q-1\}$  zufällig (Ehrlicher-Verifizierer)
3. Setze  $\tilde{a}_1 := g_1^{\tilde{b}} y_1^{\tilde{c}}$
4. Setze  $\tilde{a}_2 := g_2^{\tilde{b}} y_2^{\tilde{c}}$ . Return:  $(\tilde{a}_1, \tilde{a}_2, \tilde{c}, \tilde{b})$

Dieses Transcript ist ein akzeptierendes Transcript und alle Elemente wurden zufällig gewählt, wie die Elemente eines durch  $(P,V)$  erzeugten Transcripts. Die Eigenschaft, dass das Protokoll ein EV-ZKP ist, kann ausgenutzt werden, um den Test der Angaben der Auszähler sicher zu machen:

Umwandeln des obigen Protokolls in ein nicht-interaktives mittels einer kollisions sicheren Hashfunktion  $h : \{0, 1\}^* \rightarrow \mathbb{Z}_p$ . Damit werden nun aus  $y_1, y_2, g_1, g_2, c, b$  bestimmt, was einem ehrlichen Verifizierer entspricht. (Es gilt  $y_1 = g_1^x, y_2 = g_2^x$ )

**P**  $\rightarrow$  **V**  $r \in \{0, \dots, q-1\}$  zufällig,  $a := (a_1, a_2) := (g_1^r, g_2^r)$  und übertrage dann  $c := h(g_1 y_1 g_2 y_2 a_1 a_2)$  und  $b := r - cx$ .

**V** testet dann ob  $c \stackrel{?}{=} h(g_1 y_1 g_2 y_2 g_1^b y_1^c g_2^b y_2^c)$

Dieses Protokoll ist dann vom Sicherheitsaspekt ausreichend, da es einen EV-ZKP beinhaltet.

**Wahlsystem ohne Zentrum:** Hierfür muß das Zentrum einfach eliminiert werden und dessen Aufgaben an die  $n$  Auszähler verteilt werden. Die Erstellung eines ElGamal Verschlüsselungsverfahrens, kann wie folgt von  $A_1, A_2, \dots, A_n$  ausgeführt werden.

1. Alle  $A_i$  einigen sich auf einen Algorithmus  $A_s$  zur Schlüsselerzeugung mit Zufallseingabe  $r$
2. Jedes  $A_i$  wählt  $r_i$  und bestimmt  $c_i = \text{LockableBox}(r_i)$
3. Alle  $A_i$  veröffentlichen die  $c_i$  und anschließend die  $r_i$
4. Alle  $A_i$  testen ob  $c_j = \text{LockableBox}(r_j)$  und bestimmen  $r := \bigoplus_{i=1}^n r_i$
5. Alle  $A_i$  nutzen  $A_s$  um  $p, q, g$  zu bestimmen.

**Verteilter Aufbau von ElGamal:**

1. Jedes  $A_i$  wählt seinen geheimen Schlüssel  $x_i \in \{0, \dots, q-1\}$  und setzt  $h_i := g^{x_i}$  und  $c_i := \text{LockableBox}(h_i)$
2. Alle veröffentlichen ihre  $c_i$
3. Über geheimen Kanal öffnen alle  $A_i$  ihre  $c_i$
4. Die  $A_i$  bestimmen  $h := \sum_{i=1}^n h_i$  den gemeinsamen pub Key. Der priv Key ist dann  $x := \sum_{i=1}^n x_i$

**Verteiltes Aufbauen des  $(t, n)$ -Threshold-Scheme:** Im weiteren muß noch das Threshold-Scheme von den  $A_i$  aufgebaut werden. Idee: Jedes  $A_i$  bestimmt ein eigenes  $(t, n)$ -Threshold-Scheme welches dann zu einem gemeinsamen  $(t, n)$ -Threshold-Scheme zusammengefasst wird.

Das Geheimnis  $x$  ist dann schon an die Teilnehmer verteilt ( $x_i$ ). Idee ist nun die Koeffizienten der persönlichen Polynome an die Teilnehmer zu verteilen und somit auch die Anteile der Stützstellen der persönlichen Polynome.

1. D.h.  $A_i$  bestimmt  $f_i(X) \in \mathbb{Z}_p[X]$  ein Polynom vom Grad  $t-1$  mit  $f_i(0) = x_i$
2. Dann ergibt sich das gemeinsame Polynom  $f(X) := \sum_{i=0}^{t-1} f_i(X) | f(0) = x$  und es muß sichergestellt werden, dass  $f(X)$  vom Grad  $t-1$  ist.
3. Der Anteil von  $A_j$  ist dann  $(j, f(j)) | f(j) = \sum_{i=0}^{t-1} f_i(j)$  (Also Addition der Stützstellen)
4. Wichtig: Die Korrektheit der Rechnung muß getestet werden.

**Verteilter Aufbau des  $(t, n)$ -Threshold-Scheme:**

1. Alle  $A_i$  wählen Primzahl (groß genug)  $p > \max x, n$
2. Jedes  $A_i$  wählt  $f_{i,j} \in \{0, \dots, p-1\}$  für  $j = 1, \dots, t-1$  und setze  $f_i, 0 := x_i: f_i(X) := \sum_{j=0}^{t-1} f_{i,j} X^j$ . Also bestimmen der einzelnen Polynome.
3. Alle  $A_i$  setzen dann  $F_{i,j} := g^{f_{i,j}}$ , also verschlüsseln und dann veröffentlichen (für  $j = 1, \dots, t-1$ ). Damit sind die Koeffizienten nicht mehr veränderbar.
4. Nun muß der Grad noch getestet werden, also ob:  $\sum_{i=1}^n f_i(x)$  einen Grad von  $t-1$  hat. Dies können wir auf den verschlüsselten Werten tun: Teste:  $\prod_{i=1}^n F_{i,t-1} \stackrel{?}{\neq} 1$  Also:  $\sum_{i=1}^n f_{i,t-1} \stackrel{?}{\neq} 0$
5. Dann muß jedes  $A_i$  die  $s_{i,l} = f_i(l)$  an die anderen  $A_l$  über sichere Kanäle verteilen.
6. Jeder muß schauen ob er die richtigen empfangen hat, also ist meine Stützstelle am Ende der Wert des Polynoms: Also  $f_l(i) \stackrel{?}{=} \sum_{j=0}^{t-1} f_{l,j} i^j$ , dass kann er aber nicht. Er kann aber auf die verschlüsselten Werte zurückgreifen:  $g^{s_l, i} \stackrel{?}{=} \prod_{j=0}^{t-1} (F_{l,j})^i$
7. Jedes  $A_i$  setzt  $s_i := \sum_{l=1}^n s_{l,i}$
8. Jedes  $A_i$  signiert bei fehlerfreiem Ablauf das  $h$ .

Walter: „Die ganze Indexwuselei würde ich nicht auswendig lernen, sondern eher die Idee wissen. Was haben wir gemacht? Threshold-Scheme und Homomorphismus häufiger angewendet als uns lieb war. Und plötzlich lief das Ding.“

## 4.4 Elektronisches Geld

### 4.4.1 Einleitung

Anforderungen:

1. Soll so sicher wie eine normale Brieftasche sein
2. Sollte den selben Freiheitsgrad wie beim Umgang mit normalem Geld liefern
3. Sollte anonymen Zahlungsverkehr ermöglichen
4. Sollte keine hohen Zusatzkosten benötigen

### Einfaches Protokoll: Shimon Even 1978, deckt die ersten beiden Anforderungen ab

Grundlage ist hier ein Taschengeld mit Infrarotschnittstelle, das nicht geöffnet werden kann und den normalen, alltäglichen Geldtransfer regeln soll (Quittungen, Unterschriften).

Die Kunden A und B einer Bank wollen nun Geld von A nach B transferieren:

1. Die Bank erstellt für beide Kunden ein PK-Schlüsselpaar und schickt dieses zusammen mit dem aktuellen Kontostand, einem aktuellen Timestamp, dem öffentlichen Schlüssel der Bank und dem von der Bank unterschriebenen öffentlichen Schlüssel (= Beleg, dass der jeweilige Kunde, Kunde der Bank ist) des Kunden an diesen
2. A schickt nun seinen "ich-bin-Kunde"-Beleg zusammen mit dem zu überweisenden Betrag und seinem Timestamp an B
3. Dieser verifiziert den Betrag, kontrolliert ob der erhaltene Timestamp nicht viel von seinem abweicht und schickt den Betrag zusammen mit einer Zufallszahl, seinem "ich-bin-Kunde"-Beleg und dem von der Bank erhaltenen Timestamp zurück an A
4. A testet nun auch wieder, ob der Betrag korrekt ist und verifiziert den erhaltenen Timestamp. Danach kontrolliert A, ob er den Betrag auch überweisen kann und berechnet seinen neuen Kontostand
5. A schickt nun den Betrag, seinen Timestamp und die Zufallszahl mit seinem privaten Schlüssel verschlüsselt an B. B entschlüsselt und kontrolliert die Werte
6. B schickt nun den negativen Betrag, seinen Timestamp und die Zufallszahl mit seinem privaten Schlüssel verschlüsselt an A. Auch dieser verifiziert die Werte

Das Verfahren ist so sicher wie die benutzten Verfahren. Der Zahlungsverkehr ist nachvollziehbar. Man kann leicht pro Zahlungsverkehr Steuern erheben (niedrigerer Zinssatz möglich). Dafür wird einfach in Punkt 3 und 4 bei der Verifikation des Betrags ein  $(1 - \delta)$  für den Zinssatz  $\delta$  dazumultipliziert.

Nun soll es aber faire Systeme geben, die den Zahlungsverkehr nicht offenlegen (wer will schon, dass jemand von den Einkäufen bei Beate Uhse erfährt). Hierfür wird eine Münze blind von der Bank unterschrieben. Nur eine autorisierte Stelle (Staat) sollte den Zahlungsverkehr nachvollziehen können.

### Voraussetzungen

- $p, q$  sind Primzahlen mit  $q|p - 1$ ,  $G$  ist Untergruppe der Ordnung  $q$  in  $\mathbb{Z}^*_p$ ,  $g$  ist Generator in  $G$
- $h : \{0, 1\}^* \rightarrow \mathbb{Z}_p$  ist eine kollisions sichere Hashfunktion
- Geheimer Schlüssel:  $x \in \{0, \dots, q - 1\}$ , öffentlicher Schlüssel:  $(p, q, g, y)$  mit  $y = g^x$

**Identifikation nach Schnorr = ProofLog( $g, y$ )** Der Prover (der  $x$  von oben kennt) wählt eine Zufallszahl  $r$  und schickt  $g^r$  an den Verifier. Dieser wählt eine Zahl  $c \in \{0, \dots, q - 1\}$  und schickt diese rüber. Der Prover rechnet nun  $r - cx$  und schickt das Ergebnis an den Verifier. Dieser überprüft nun, ob folgende Gleichung gilt:  $g^r \equiv g^{r-cx} \cdot y^c (= g^{r-cx} \cdot g^{xc})$

**Identifikation nach Schnorr (nicht-interaktiv) = ProofLog<sub>h</sub>( $m, g, y$ )** Der Prover wählt wieder eine Zufallszahl  $r$  und rechnet wieder  $g^r =: a$ . Nun berechnet er  $c := h(m \circ a)$  und daraus dann  $b := r - cx$ . Die Werte  $c, b$ , und  $m$  schickt er nun an den Verifier. Dieser testet, ob  $c \equiv h(m \circ g^b y^c)$  gilt. Ist  $m = \epsilon$ , so ist dies ein nicht-interaktiver Beweis dafür, dass der Verifier  $\log_g(y)$  kennt.

Der Kunde holt nun eine auf diesem Protokoll basierende blinde Unterschrift von Bank. Diese wird transformiert im Geschäft vorgelegt, so dass es der Bank nicht möglich ist, die Unterschrift zurückzuverfolgen.

**Idee einer blinden Unterschrift** Es wird ProofLog( $g, y$ ) verwendet, nur mit der Änderung, dass der Verifier statt einem zufälligen  $c$  die zu unterschreibende Nachricht mit der Funktion  $h$  hashed.

Wenn die Bank das Transscript  $\tau' = (a' = g^r, c' = h(m), b' = r - c'x)$  speichert, so kann sie auf die Herkunft einer eingezahlten Münze schließen. Der Kunde ändert nun also  $\tau'$  um in  $\tau$ , damit seine er nicht mit seiner Transaktion von der Bank in Verbindung gebracht werden kann. Dieses  $\tau$  soll aber auch eine gültige Unterschrift sein und nur mit Hilfe von  $\tau'$  erzeugt werden können. Dies geschieht mit einer Funktion  $\text{beta} : (a', c', b') \rightarrow (a, c, b)$  mit  $a = a'^u g^v y^w$ ,  $c = uc' + w$  und  $b = ub' + v$ . Damit ist  $\tau$  auch ein gültiges Transscript einer Unterschrift, bei dem die Bank nicht mehr die Herkunft

nachvollziehen kann weil es zu jedem Transscript der Bank  $q^2$  mögliche Transscripte für den Kunden gibt. Die Bank kann auch sicher sein, dass der Kunde nicht irgendein Transscript erzeugt, das die Bank als echt ansieht. Da gibts dann nen schönen Beweis, der zeigt, dass die Bank und der Kunde den diskreten Logarithmus lösen könnten, wenn es eine weitere Möglichkeit geben würde.

**Blinde Unterschrift** =  $BlindLogSig_h(m)$  Dies ist eine Umformung des  $ProofLog(g, y)$ -Protokolls. Während der Übertragung des Transscriptes baut der Kunde daraus sein transformiertes Transscript auf.

1. Der Prover sucht sich wieder ein  $r'$  und schickt  $a' = g^{r'}$  an den Verifier
2. Dieser sucht sich  $u, v, w \in \{0, \dots, q-1\}, v \neq 0$  aus und berechnet sein  $a$  wie in 4.4.1 und sein  $c = h(m \circ a)$ . Danach berechnet er noch ein  $c' = (c - w)u^{-1}$  (Umformung nach  $c'$  aus 4.4.1) und schickt es rüber
3. Der Prover rechnet nun wieder  $b' = r' - c'x$  und schickt es rüber
4. Der Verifier testet nun wie bei  $ProofLog$ , ob die Werte korrekt sind und berechnet sein  $b = ub' + v$  und  $ProofLog_h(m, g, y) = (c, b)$
5. Die Unterschrift kann durch die Validierung der Gleichung  $c = h(m \circ g^b y^c)$  bestätigt werden

**Blinde Unterschrift darunter, dass zwei diskrete Logarithmen gleich sind** =  $BlindLogSigEq_h(g, y, m)$

1. V sucht sich ein  $s \in \{1, \dots, q-1\}$ , rechnet  $m' = m^{1/s}$  und schickt  $m'$  an P
2. P wählt sich ein  $r'$ , berechnet ein  $z' = m'^x$  sowie das übliche  $a'$  aber jetzt mit  $a' = (a'_1, a'_2) = (g^{r'}, m'^{r'})$  und schickt  $a', z'$  an V
3. V wählt nun seine  $u, v, w$ , berechnet damit  $a = (a_1, a_2) = (a_1^u g^v y^w, (a_2^u m'^v z'^w)^s)$ , sein  $z = z'^s y^t$ , sein  $c = h(m \circ z \circ a_1 \circ a_2)$  sowie daraus  $c' = (c - w)u^{-1}$  und schickt  $c'$  an P
4. P berechnet nun, wie sonst auch,  $b' = r' - c'x$  und schickt es zurück
5. V testet nun, ob  $(a'_1, a'_2) \equiv (g^{b'} y^{c'}, m'^{b'} z'^{c'})$  gilt
6. Die Unterschrift ist dann das Tripel  $(z, c, b)$ . Getestet werden kann diese durch Validierung der Gleichung  $c = h(m \circ z \circ g^b y^c \circ m^b z^c)$

#### Aufbau

- Geld entspricht einer Münze
- $q, p$  Primzahlen,  $q|p-1$
- $g, g_1, g_2$  Generatoren ( $g$  von der Bank,  $g_2$  vom Staat,  $g_1$  von den Kunden und Geschäften)
- $h$  kollisionssichere Hashfunktion

Jeder Teilnehmer (Bank, Staat, Kunde Geschäft) sucht sich nun einen geheimen Schlüssel  $x$ , setzt  $y = [\text{sein Generator}]^x$  und veröffentlicht  $y$ .

#### 4.4.2 Online Systeme

Beim Abheben einer Münze findet eine Kommunikation zwischen Bank und Staat statt. Beim Bezahlen findet eine Kommunikation zwischen Geschäft und Bank statt

### Münze von der Bank abheben, Bank führt mit dem Kunden ein $BlindLogSig_n$ mit $m = \epsilon$ durch

1. Die Bank (B) schickt  $a' = g^{r'}$  mit  $r =$  Zufallszahl an den Kunden (K)
2. Dieser berechnet sein  $a$  mit  $u, v, w$  zufällig laut 4.4.1, daraus  $c = h(a)$  und wieder  $c' = (c - w)u^{-1}$  und schickt  $c'$  und  $E_{\text{Staat}}(u, v, w)$  an B
3. B sendet die beiden Werte zusammen mit  $a'$  an den Staat (S)
4. S entschlüsselt  $E_{\text{Staat}}(u, v, w)$  und testet, ob K das  $c'$  richtig berechnet hat. Danach speichert er  $(a', c') \leftrightarrow (u, v, w)$  und gibt ggf. der Bank das ok
5. Bekommt die Bank das ok vom Staat, berechnet sie das  $b' = r' - c'x$  und schickt es an K
6. K testet das  $b'$  durch Überprüfung, ob  $a' \equiv g^{b'}y^{c'}$  gilt und berechnet daraus nun sein  $b = ub' + v$ . Seine Münze ist nun  $(c, b)$

**Geld ausgeben** Der Kunde K gibt seine Münze  $(c, b)$  an das Geschäft G. Dieses testet, ob  $c = h(g^b y^c)$  gilt und schickt das Tupel auch an die Bank B. Diese prüft  $(c, b)$  nochmal auf die Gleiche Weise und kontrolliert, ob die Münze schonmal eingezahlt wurde. Wenn nicht, gibt sie das ok für die Transaktion.

### 4.4.3 Offline Systeme

### 4.5 Broadcast

**Einleitung:** Verschlüsseltes Senden der selben Nachricht an viele Teilnehmer. Dabei soll es jeder Teilmenge  $S$  von Teilnehmern möglich sein die Nachricht zu entschlüsseln. Auch soll es Teilnehmern die nicht aus  $S$  sind nicht möglich sein die Nachricht zu entschlüsseln, nicht mal, wenn sie kooperieren.

Anwendungen sind Pay-TV, Verschlüsselte Dateisysteme und DVDs.

Im folgenden:  $n$  potentielle Teilnehmer mit  $[n] = \{1, \dots, n\}$ .  $S \subseteq [n]$  berechtigten Teilnehmer und  $M$  die Nachricht. Naiver Ansatz:

- Jeder Teilnehmer  $i \in [n]$  erzeugt Schlüsselpaar  $(E_i, D_i)$  und veröffentlicht  $E_i$
- Sender erzeugt zufälligen Schlüssel  $K$  für ein symmetrisches Verschlüsselungsverfahren.
- BROADCAST:  $\forall i \in S : E_i(K) \wedge E_k(M)$

Zusätzliche Information hat Größenordnung  $\Theta(|S|)$

### Aufbau

SETUP( $n$ )  $\rightarrow$  ( $PK, d_1, \dots, d_n$ )

**Eingabe:**  $n$  Anzahl potentieller Teilnehmer

**Ausgabe:** öffentlicher Schlüssel  $PK$ , privater Schlüssel  $d_1, \dots, d_n$

Wähle  $g \in G$  und  $\alpha, \gamma \in \{0, \dots, p-1\}$  zufällig, setze für  $g_i = g^{\alpha_i}$ :  $PK = (g, g_1, \dots, g_n, g_{n+2}, \dots, g_{2n}, v = g^\gamma)$

Private Schlüssel:  $d_i = g_i^\gamma = v^{\alpha_i}$

ENCRYPT( $S, PK$ )  $\rightarrow$  ( $Hdr, K$ )

**Eingabe:**  $S \subseteq [n]$  Menge berechtigter Teilnehmer,  $PK$

**Ausgabe:**  $Hdr$  Header,  $K$  symmetrischer Schlüssel

**BROADCAST:** ( $S, Hdr$ ) und mit  $K$  verschlüsselte Nachricht

Wähle  $t \in \{0, \dots, p-1\}$  zufällig und setze  $K = e(g_{n+1}, g)^t = e(g_n, g_1)^t$ . Sowie  $Hdr = (g^t, (v \cdot \prod_{j \in S} g_{n+1-j})^t)$ .

DECRYPT( $S, i, d_i, Hdr, PK$ )  $\rightarrow$   $K$

**Eingabe:**  $S, i \in S, d_i, Hdr, PK$

**Ausgabe:** symmetrischer Schlüssel  $K$

Bekannt sind:  $PK = (g, \dots, g_{2n}, v = g^\gamma)$  sowie  $Hdr = (g^t, (v \cdot \prod_{j \in S} g_{n+1-j})^t), d_i = v^{\alpha^i}$

Gesucht:  $K = e(g_{n+1}, g)^t$

$e(g_i, (v \cdot \prod_{j \in S} g_{n+1-j})^t) = e(g_i, g_{n+1-i}^t) \cdot e(g_i, (v \cdot \prod_{j \in S: j \neq i} g_{n+1-j})^t) = e(g, g_{n+1})^t \cdot e(g, v^{\alpha^i} \cdot \prod_{j \in S: j \neq i} g_{n+1-j+i})^t$

und setze  $K = e(g_i, (v \cdot \prod_{j \in S} g_{n+1-j})^t) \cdot e(g, v^{\alpha^i} \cdot \prod_{j \in S: j \neq i} g_{n+1-j+i})^{-t}$

•  $SETUP(n) = (PK, d_1, \dots, d_N)$  und  $ENCRYPT(S, PK) = (Hdr, K) \Rightarrow \forall i \in S : DECRYPT(S, i, d_i, Hdr, PK) = K$

**Bilineare Abbildungen:**  $G, G'$  zyklische Gruppen der Ordnung  $p$  ( $p$  Primzahl).  $g$  Generator von  $G$ , dann heißt die Abbildung  $e : G \times G \rightarrow G'$  bilinear wenn gilt:

1. Bilinear:  $\forall u, v \in G, \forall a, b \in \mathbb{Z} : e(u^a, v^b) = e(u, v)^{ab}$
2. Nicht degeneriert:  $e(g, g) \neq 1$
3. Effizient:  $e$  muß effizient berechenbar sein

Mit folgenden Eigenschaften:

- symmetrisch:  $e(u, v) = e(v, u)$
- distributiv:  $e(u, vv') = e(u, v) \cdot e(u, v')$
- $e(u^a, v) = e(u, v^a)$

**Verallgemeinertes Diffie-Hellman Problem:** Sei  $G$  zyklische Gruppe der Ordnung  $p$ ,  $g$  und  $h$  Generatoren von  $G$ .  $e : G \times G \rightarrow G'$  eine bilineare Abbildung.  $\alpha \in \{0, \dots, p-1\}$  zufällig gewählt, dann:

### I-BDHE

Gegeben:  $(g, g^\alpha, g^{\alpha^2}, \dots, g^{\alpha^l}, g^{\alpha^{l+2}}, \dots, g^2) \in G^{2l+1}$

Gesucht:  $e(g, h)^{\alpha^{l+1}} = e(g^{\alpha^{l+1}}, h)$

Kein effizienter Algorithmus zur Lösung von I-BDHE bekannt.

**Effizienz:** Zeit zum Entschlüsseln wird durch  $W_S^{(i)} = v^{\alpha^i} \cdot \prod_{j \in S: j \neq i} g_{n+1-j+i}$  dominiert wofür man  $|S| - 2$  Gruppenoperationen braucht. Ist  $W_S^{(i)}$  bekannt, so kann  $W_{S'}^{(i)}$  mit  $|S \setminus S'| + |S' \setminus S|$  Gruppenoperationen berechnet werden. Der geheime Schlüssel  $d_i$  kann zusätzlich auch noch  $W_{[n]}^{(i)}$  enthalten. Nützlich falls  $|S| = n - r$  für kleines  $r$ .

**Eigenschaften des einfachen Protokolls:**

- Größe des Public Keys:  $2n + 1$  Gruppenelemente
- Größe des Headers:  $2$  Gruppenelemente ( $+\Theta(|S| \log n)$ ) Bits für Kodierung von  $S$ ).
- Größe der privaten Schlüssel:  $1$  Gruppenelement.

Beispiel verschlüsseltes Dateisystem: Bei naivem Ansatz mit vernünftigen Parametern ca.  $44|S|$  Bytes, beim einfachen Protokoll  $4|S|$  Bytes +  $2$  Gruppenelemente (32-Bit lange Benutzer-IDs)

**Verbessertes Protokoll:** Benutze  $A = \lceil \frac{n}{B} \rceil$  Instanzen für jeweils  $B$  Teilnehmer und teile Informationen zwischen den Instanzen. Neues  $SETUP_B$ ,  $ENCRYPT$  und  $DECRYPT$  funktioniert analog wie im einfachen Protokoll.

Eigenschaften des verbesserten Protokolls für  $A = B = \sqrt{n}$

- Größe des Public Keys:  $2B + A = \Theta(\sqrt{n})$  Gruppenelemente.
- Größe des Headers:  $A + 1 = \Theta(\sqrt{n})$  Gruppenelemente (+ Platz für Kodierung von  $S$ ).
- Größe der privaten Schlüssel: ein Gruppenelement.

Anwendung: Sperren von DVD Player bei zukünftigen DVDs. Platzbedarf bei realistischen Parametern ca.  $5MB + 4r$  Bytes mit  $r$  Anzahl gesperrter DVD Player.

**Sicherheit:**  $\mathcal{A}$  Angreifer,  $\mathcal{C}$  Challenger. Modell:

**Init**  $\mathcal{A}$  wählt  $S^* \subseteq [n]$  = nicht autorisierte Teilnehmer.

**Setup**  $\text{SETUP}(n)$  wird aufgerufen.  $\mathcal{A}$  erhält Public Key  $PK$  und Private Key  $d_i | i \in S^*$

**Challenge**  $\mathcal{C}$  führt  $\text{ENCRYPT}$  durch und erhält  $(Hdr, K)$ .  $\mathcal{C}$  wählt  $b \in \{0, 1\}$  zufällig und setzt  $K_b = K$  und wählt  $K_{1-b}$  zufällig.  $\mathcal{C}$  schickt  $(Hdr, K_0, K_1)$  an  $\mathcal{A}$

**Guess**  $\mathcal{A}$  wählt  $b' \in \{0, 1\}$ . Er gewinnt, falls  $b = b'$  gilt.

**Definition.** Das Broadcasting-System ist  $(t, \epsilon, n)$ -sicher, wenn für jeden Algorithmus  $\mathcal{A}$ , der höchstens  $t$  Rechenschritte läuft, gilt  $\Pr[b = b'] < \frac{1}{2} + \epsilon$

**Theorem.** Es gibt ein Polynom  $p$ , so dass für alle  $t, \epsilon, n$  gilt: Wenn das  $n$ -BDHE  $(t, \epsilon, n)$ -sicher ist, dann ist das Broadcasting-System  $(p(n)t, \epsilon, n)$ -sicher.

**Zusammenfassung:**

- $|Hdr| + |PK| = \Theta(\sqrt{n})$
- So sicher wie  $n$ -BDHE

Verallgemeinerung und offene Fragen:

- Es gibt Variante des Protokolls, die sicher bzgl. Chosen-Ciphertext-Angriffen ist.
- Es gibt Variante des Protokolls mit Traitor Tracing.
- Gibt es ein Verfahren mit kurzen privaten Schlüsseln, bei dem die Größe von Header und public Key kleiner als  $\Omega(\sqrt{n})$  ist?

## 5 Fragen

Welche klassischen Verfahren gibt es?

Warum heißen die klassischen Verfahren auch symmetrischen Verfahren?

Was ist die Idee beim C-36 Verfahren?

Was ist die Idee beim DES Verfahren?

Welche Verbesserungen von DES gibt es?

Was ist die Idee beim IDEA Verfahren?

Was ist die Idee beim AES Verfahren?

Wie wird bei IDEA entschlüsselt?

Wie wird bei DES entschlüsselt?

Wann ist ein Verfahren sicher?

## 5.1 Public Key

Idee von PK?

Unterschied zu klassischen Verfahren?

Idee bei Unterschriften?

Wie Unterschriften mit klassischen Verfahren?

Braucht man PK-Verfahren um Nachricht auszutauschen ohne sich vorher zu treffen?

Wie wird das Rucksackverfahren aufgebaut?

Wie erfolgt der Angriff?

## 5.2 ElGamal

Was wäre wenn  $g$  kein Generator wäre?

Was ist die Idee des Verfahrens nach ElGamal?

Wie sicher ist das Verfahren nach ElGamal?

Wie macht man eine Unterschrift nach ElGamal?

## 5.3 Elliptische Kurven

Was ist der Unterschied zwischen der Verschlüsselung mit Elliptischen Kurven und ElGamal?

Was ist die Idee bei der Verschlüsselung mit Elliptischen Kurven?

## 5.4 Einfache Protokolle

Welche Arten der vermesslichen Übertragung gibt es?

Welche Protokolle zur vermesslichen Übertragung gibt es?

Welche Protokolle zum Bit Commitment gibt es?

Ist gleichzeitige perfekte Sicherheit für beide Parteien beim Bit Commitment möglich?

Wie geht das Identifikationsverfahren von Shamir?

## 5.5 Zero-Knowledge-Proof

Wie ist die formale Definition eines ZKP?

Wie geht ein ZKP für Graphenisomorphismus?

Independent Set?

Hamilton Kreis?

3-Färbung?

3-SAT?

Wie geht der ZKP nach Shamir?

Wie geht der ZKP nach Fiat-Shamir?

Für welche Probleme aus NPC gibt es ZKP?

Ist die Parallelausführung eines ZKP auch wieder ZKP?

Ist die Seriellausführung eines ZKP auch wieder ZKP?

Wie macht man eine Unterschrift mit ZKP als Basis?

## 5.6 Elektronische Wahlen I

Welche einfachen Wahlprotokolle gibt es?

Wie wird typischerweise mit mehreren Mischern gemischt?

Was ist die Idee beim ersten Protokoll mit Mischern?

Wozu dient die Funktion  $\phi$  beim Wählen mit Mischern?

Welche Idee wird benutzt, um eine Wahl zu machen, bei der die Wähler nicht erpressbar sind?

## 5.7 Elektronische Wahl II

Wozu dient eine Homomorphe Verschlüsselung?

Wozu dient ein  $(t, n)$ -Threshold-Scheme?

Wie baut man ein  $(t, n)$ -Threshold-Scheme?

Was ist die Idee des obigen Wahlverfahrens?

Wie ist die Idee des gemeinsamen Aufbau eines Verschlüsselungsverfahrens?

Was ist die Idee des gemeinsamen Aufbau eines  $(t, n)$ -Threshold-Scheme?