Wurstkessel

Kryptographie in Forth

Bernd Paysan

Forth–Tagung 2009

Übersicht

Motivation

- MD5 und SHA–1 sind (mehr oder weniger) geknackt $\longrightarrow$ Ersatz ist nötig
- Inspiration: Tabellengetriebene Implementierung von „Whirlpool“
- Einfach in Hard- und Software implementierbar (Ziel: 64–Bit–CPUs)
- Hoher Durchsatz zumindest theoretisch möglich ($\longrightarrow$ Parallelität)
- Grundsätzlich skalierbar (32 Bit: Kleinere Hashes, kürzere Schlüssel)

Motivation

- MD5 und SHA–1 sind (mehr oder weniger) geknackt $\longrightarrow$ Ersatz ist nötig
- Inspiration: Tabellengetriebene Implementierung von „Whirlpool“
- Einfach in Hard- und Software implementierbar (Ziel: 64–Bit–CPUs)
- Hoher Durchsatz zumindest theoretisch möglich ($\longrightarrow$ Parallelität)
- Grundsätzlich skalierbar (32 Bit: Kleinere Hashes, kürzere Schlüssel)

Motivation

- MD5 und SHA–1 sind (mehr oder weniger) geknackt $\longrightarrow$ Ersatz ist nötig
- Inspiration: Tabellengetriebene Implementierung von „Whirlpool“
- Einfach in Hard- und Software implementierbar (Ziel: 64–Bit–CPUs)
- Hoher Durchsatz zumindest theoretisch möglich ($\longrightarrow$ Parallelität)
- Grundsätzlich skalierbar (32 Bit: Kleinere Hashes, kürzere Schlüssel)

Motivation

- MD5 und SHA–1 sind (mehr oder weniger) geknackt $\longrightarrow$ Ersatz ist nötig
- Inspiration: Tabellengetriebene Implementierung von „Whirlpool“
- Einfach in Hard- und Software implementierbar (Ziel: 64–Bit–CPUs)
- Hoher Durchsatz zumindest theoretisch möglich ($\longrightarrow$ Parallelität)
- Grundsätzlich skalierbar (32 Bit: Kleinere Hashes, kürzere Schlüssel)

Motivation

- MD5 und SHA–1 sind (mehr oder weniger) geknackt $\longrightarrow$ Ersatz ist nötig
- Inspiration: Tabellengetriebene Implementierung von „Whirlpool“
- Einfach in Hard- und Software implementierbar (Ziel: 64–Bit–CPUs)
- Hoher Durchsatz zumindest theoretisch möglich ($\longrightarrow$ Parallelität)
- Grundsätzlich skalierbar (32 Bit: Kleinere Hashes, kürzere Schlüssel)

Kryptographische Grundlagen

Asymmetrische Kryptographie

- Schlüsselaustausch: öffentlicher/privater Schlüssel
- Digitale Signatur
- Verschlüsselung nur von Hashes und symmetrischen Schlüsseln, weil zu langsam
- Beispiele: RSA, Ellyptic Keys

Symmetrische Kryptographie

- Eigentliche Verschlüsselung der Nachricht
- Sichere Hashes
- (Pseudo–)Zufallszahlen
- Beispiele: AES, SHA–1

Kryptographische Grundlagen

Asymmetrische Kryptographie

- Schlüsselaustausch: öffentlicher/privater Schlüssel
- Digitale Signatur
- Verschlüsselung nur von Hashes und symmetrischen Schlüsseln, weil zu langsam
- Beispiele: RSA, Ellyptic Keys

Symmetrische Kryptographie

- Eigentliche Verschlüsselung der Nachricht
- Sichere Hashes
- (Pseudo–)Zufallszahlen
- Beispiele: AES, SHA–1

Kryptographische Grundlagen

Asymmetrische Kryptographie

- Schlüsselaustausch: öffentlicher/privater Schlüssel
- Digitale Signatur
- Verschlüsselung nur von Hashes und symmetrischen Schlüsseln, weil zu langsam
- Beispiele: RSA, Ellyptic Keys

Symmetrische Kryptographie

- Eigentliche Verschlüsselung der Nachricht
- Sichere Hashes
- (Pseudo–)Zufallszahlen
- Beispiele: AES, SHA–1

Kryptographische Grundlagen

Asymmetrische Kryptographie

- Schlüsselaustausch: öffentlicher/privater Schlüssel
- Digitale Signatur
- Verschlüsselung nur von Hashes und symmetrischen Schlüsseln, weil zu langsam
- Beispiele: RSA, Ellyptic Keys

Symmetrische Kryptographie

- Eigentliche Verschlüsselung der Nachricht
- Sichere Hashes
- (Pseudo–)Zufallszahlen
- Beispiele: AES, SHA–1

Kryptographische Grundlagen

Asymmetrische Kryptographie

- Schlüsselaustausch: öffentlicher/privater Schlüssel
- Digitale Signatur
- Verschlüsselung nur von Hashes und symmetrischen Schlüsseln, weil zu langsam
- Beispiele: RSA, Ellyptic Keys

Symmetrische Kryptographie

- Eigentliche Verschlüsselung der Nachricht
- Sichere Hashes
- (Pseudo–)Zufallszahlen
- Beispiele: AES, SHA–1

Kryptographische Grundlagen

Asymmetrische Kryptographie

- Schlüsselaustausch: öffentlicher/privater Schlüssel
- Digitale Signatur
- Verschlüsselung nur von Hashes und symmetrischen Schlüsseln, weil zu langsam
- Beispiele: RSA, Ellyptic Keys

Symmetrische Kryptographie

- Eigentliche Verschlüsselung der Nachricht
- Sichere Hashes
- (Pseudo–)Zufallszahlen
- Beispiele: AES, SHA–1

Kryptographische Grundlagen

Asymmetrische Kryptographie

- Schlüsselaustausch: öffentlicher/privater Schlüssel
- Digitale Signatur
- Verschlüsselung nur von Hashes und symmetrischen Schlüsseln, weil zu langsam
- Beispiele: RSA, Ellyptic Keys

Symmetrische Kryptographie

- Eigentliche Verschlüsselung der Nachricht
- Sichere Hashes
- (Pseudo–)Zufallszahlen
- Beispiele: AES, SHA–1

Verschlüsselung=Zufall

- Alle drei Anwendungen symmetrischer Verfahren können auf derselben Funktion basieren
- Gute Verschlüsselung ist möglichst ununterscheidbar von Zufall
- Ein Schlüssel darf nur durch brutales Ausprobieren gefunden werden
- Zwischen Text und Hash darf ebenfalls keine erkennbare Beziehung stehen
  - Vom Hash darf nicht auf die Nachricht geschlossen werden
  - Ein gezieltes Manipulieren der Nachricht darf nicht zu einem vorher festgelegten Hashwert führen

Verschlüsselung=Zufall

- Alle drei Anwendungen symmetrischer Verfahren können auf derselben Funktion basieren
- Gute Verschlüsselung ist möglichst ununterscheidbar von Zufall
- Ein Schlüssel darf nur durch brutales Ausprobieren gefunden werden
- Zwischen Text und Hash darf ebenfalls keine erkennbare Beziehung stehen
  - Vom Hash darf nicht auf die Nachricht geschlossen werden
  - Ein gezieltes Manipulieren der Nachricht darf nicht zu einem vorher festgelegten Hashwert führen

Verschlüsselung=Zufall

- Alle drei Anwendungen symmetrischer Verfahren können auf derselben Funktion basieren
- Gute Verschlüsselung ist möglichst ununterscheidbar von Zufall
- Ein Schlüssel darf nur durch brutales Ausprobieren gefunden werden
- Zwischen Text und Hash darf ebenfalls keine erkennbare Beziehung stehen
  - Vom Hash darf nicht auf die Nachricht geschlossen werden
  - Ein gezieltes Manipulieren der Nachricht darf nicht zu einem vorher festgelegten Hashwert führen

Verschlüsselung=Zufall

- Alle drei Anwendungen symmetrischer Verfahren können auf derselben Funktion basieren
- Gute Verschlüsselung ist möglichst ununterscheidbar von Zufall
- Ein Schlüssel darf nur durch brutales Ausprobieren gefunden werden
- Zwischen Text und Hash darf ebenfalls keine erkennbare Beziehung stehen
  - Vom Hash darf nicht auf die Nachricht geschlossen werden
  - Ein gezieltes Manipulieren der Nachricht darf nicht zu einem vorher festgelegten Hashwert führen

Verschlüsselung=Zufall

- Alle drei Anwendungen symmetrischer Verfahren können auf derselben Funktion basieren
- Gute Verschlüsselung ist möglichst ununterscheidbar von Zufall
- Ein Schlüssel darf nur durch brutales Ausprobieren gefunden werden
- Zwischen Text und Hash darf ebenfalls keine erkennbare Beziehung stehen
  - Vom Hash darf nicht auf die Nachricht geschlossen werden
  - Ein gezieltes Manipulieren der Nachricht darf nicht zu einem vorher festgelegten Hashwert führen

Verschlüsselung=Zufall

- Alle drei Anwendungen symmetrischer Verfahren können auf derselben Funktion basieren
- Gute Verschlüsselung ist möglichst ununterscheidbar von Zufall
- Ein Schlüssel darf nur durch brutales Ausprobieren gefunden werden
- Zwischen Text und Hash darf ebenfalls keine erkennbare Beziehung stehen
  - Vom Hash darf nicht auf die Nachricht geschlossen werden
  - Ein gezieltes Manipulieren der Nachricht darf nicht zu einem vorher festgelegten Hashwert führen

Der Fleischwolf

Die Funktion ϕ dreht einen Block Daten zusammen mit einem Schlüssel durch den Wolf, und ist damit der Kern jeder Ver- und Entschlüsselung:

$$\phi(a, s, e) \rightarrow a', s', e'$$

- a Der Akkumulator für den Hash–Wert
- s Der Schlüssel (ändert sich während der Verschlüsselung)
- e Die Nachricht (liefert Entropie)

ϕ muss einfach zu berechnen sein, das Ergebnis die Ausgangsdaten aber irreversibel verändern

Der Fleischwolf

Die Funktion ϕ dreht einen Block Daten zusammen mit einem Schlüssel durch den Wolf, und ist damit der Kern jeder Ver- und Entschlüsselung:

$$\phi(a, s, e) \rightarrow a', s', e'$$

- a Der Akkumulator für den Hash–Wert
- s Der Schlüssel (ändert sich während der Verschlüsselung)
- e Die Nachricht (liefert Entropie)

ϕ muss einfach zu berechnen sein, das Ergebnis die Ausgangsdaten aber irreversibel verändern

Der Fleischwolf

Die Funktion ϕ dreht einen Block Daten zusammen mit einem Schlüssel durch den Wolf, und ist damit der Kern jeder Ver- und Entschlüsselung:

$$\phi(a, s, e) \rightarrow a', s', e'$$

- a Der Akkumulator für den Hash–Wert
- s Der Schlüssel (ändert sich während der Verschlüsselung)
- e Die Nachricht (liefert Entropie)

ϕ muss einfach zu berechnen sein, das Ergebnis die Ausgangsdaten aber irreversibel verändern

Der Fleischwolf

Die Funktion ϕ dreht einen Block Daten zusammen mit einem Schlüssel durch den Wolf, und ist damit der Kern jeder Ver- und Entschlüsselung:

$$\phi(a, s, e) \rightarrow a', s', e'$$

- a Der Akkumulator für den Hash–Wert
- s Der Schlüssel (ändert sich während der Verschlüsselung)
- e Die Nachricht (liefert Entropie)

ϕ muss einfach zu berechnen sein, das Ergebnis die Ausgangsdaten aber irreversibel verändern

Der Fleischwolf

Die Funktion ϕ dreht einen Block Daten zusammen mit einem Schlüssel durch den Wolf, und ist damit der Kern jeder Ver- und Entschlüsselung:

$$\phi(a, s, e) \rightarrow a', s', e'$$

- a Der Akkumulator für den Hash–Wert
- s Der Schlüssel (ändert sich während der Verschlüsselung)
- e Die Nachricht (liefert Entropie)

ϕ muss einfach zu berechnen sein, das Ergebnis die Ausgangsdaten aber irreversibel verändern

Der Algorithmus

rng_x 256 64–bittige Zufallszahlen von `www.random.org` in einer Tabelle

$rol(rng_x, 1) \oplus rng_y$ gibt 64k zufällig aussehende Zahlen

$r^j = \bigoplus_{i=0}^{7} rol(rng_{x_i^j}, i)$ gibt ca. $2^{64} - 2^{32}$ zufällig aussehende Zahlen

$x_i^j = a_{p(8j+i)} \oplus s_j^i$ Index in die Tabelle: Bytes aus dem Akkumulator (in verschiedenen Schrittweiten durchlaufen) mit einem Byte des transponierten Status verknüpfen

$s^j = s^{p(j)} \oplus r^j$ Die 64–Bit–Worte des Status werden permutiert und mit den generierten Zahlen verXODERt.

$e' = a \oplus e, a' = s \oplus e'$ Akkumulator, Message („Entropie“) und Status miteinander verXODERn.

Der Algorithmus

rng_x 256 64–bittige Zufallszahlen von `www.random.org` in einer Tabelle

$rol(rng_x, 1) \oplus rng_y$ gibt 64k zufällig aussehende Zahlen

$r^j = \bigoplus_{i=0}^{7} rol(rng_{x_i^j}, i)$ gibt ca. $2^{64} - 2^{32}$ zufällig aussehende Zahlen

$x_i^j = a_{p(8j+i)} \oplus s_j^i$ Index in die Tabelle: Bytes aus dem Akkumulator (in verschiedenen Schrittweiten durchlaufen) mit einem Byte des transponierten Status verknüpfen

$s^j = s^{p(j)} \oplus r^j$ Die 64–Bit–Worte des Status werden permutiert und mit den generierten Zahlen verXODERt.

$e' = a \oplus e, a' = s \oplus e'$ Akkumulator, Message („Entropie“) und Status miteinander verXODERn.

Der Algorithmus

rng_x 256 64–bittige Zufallszahlen von `www.random.org` in einer Tabelle

$rol(rng_x, 1) \oplus rng_y$ gibt 64k zufällig aussehende Zahlen

$r^j = \bigoplus_{i=0}^{7} rol(rng_{x_i^j}, i)$ gibt ca. $2^{64} - 2^{32}$ zufällig aussehende Zahlen

$x_i^j = a_{p(8j+i)} \oplus s_j^i$ Index in die Tabelle: Bytes aus dem Akkumulator (in verschiedenen Schrittweiten durchlaufen) mit einem Byte des transponierten Status verknüpfen

$s^j = s^{p(j)} \oplus r^j$ Die 64–Bit–Worte des Status werden permutiert und mit den generierten Zahlen verXODERt.

$e' = a \oplus e, a' = s \oplus e'$ Akkumulator, Message („Entropie“) und Status miteinander verXODERn.

Der Algorithmus

rng_x 256 64–bittige Zufallszahlen von `www.random.org` in einer Tabelle

$rol(rng_x, 1) \oplus rng_y$ gibt 64k zufällig aussehende Zahlen

$r^j = \bigoplus_{i=0}^{7} rol(rng_{x_i^j}, i)$ gibt ca. $2^{64} - 2^{32}$ zufällig aussehende Zahlen

$x_i^j = a_{p(8j+i)} \oplus s_j^i$ Index in die Tabelle: Bytes aus dem Akkumulator (in verschiedenen Schrittweiten durchlaufen) mit einem Byte des transponierten Status verknüpfen

$s^j = s^{p(j)} \oplus r^j$ Die 64–Bit–Worte des Status werden permutiert und mit den generierten Zahlen verXODERt.

$e' = a \oplus e, a' = s \oplus e'$ Akkumulator, Message („Entropie“) und Status miteinander verXODERn.

Der Algorithmus

rng_x 256 64–bittige Zufallszahlen von `www.random.org` in einer Tabelle

$rol(rng_x, 1) \oplus rng_y$ gibt 64k zufällig aussehende Zahlen

$r^j = \bigoplus_{i=0}^{7} rol(rng_{x_i^j}, i)$ gibt ca. $2^{64} - 2^{32}$ zufällig aussehende Zahlen

$x_i^j = a_{p(8j+i)} \oplus s_j^i$ Index in die Tabelle: Bytes aus dem Akkumulator (in verschiedenen Schrittweiten durchlaufen) mit einem Byte des transponierten Status verknüpfen

$s^j = s^{p(j)} \oplus r^j$ Die 64–Bit–Worte des Status werden permutiert und mit den generierten Zahlen verXODERt.

$e' = a \oplus e, a' = s \oplus e'$ Akkumulator, Message („Entropie“) und Status miteinander verXODERn.

Der Algorithmus

rng_x 256 64–bittige Zufallszahlen von `www.random.org` in einer Tabelle

$rol(rng_x, 1) \oplus rng_y$ gibt 64k zufällig aussehende Zahlen

$r^j = \bigoplus_{i=0}^{7} rol(rng_{x_i^j}, i)$ gibt ca. $2^{64} - 2^{32}$ zufällig aussehende Zahlen

$x_i^j = a_{p(8j+i)} \oplus s_j^i$ Index in die Tabelle: Bytes aus dem Akkumulator (in verschiedenen Schrittweiten durchlaufen) mit einem Byte des transponierten Status verknüpfen

$s^j = s^{p(j)} \oplus r^j$ Die 64–Bit–Worte des Status werden permutiert und mit den generierten Zahlen verXODERt.

$e' = a \oplus e, a' = s \oplus e'$ Akkumulator, Message („Entropie“) und Status miteinander verXODERn.

Naive Implementierung

```
8table round# 13 , 29 , 19 , 23 , 31 , 47 , 17 , 37 ,
8table permut# 2 , 6 , 1 , 4 , 7 , 0 , 5 , 3 ,
\ permut length 15
: mix2bytes ( index n k -- b1 .. b8 index' n )
    wurst-state + 8 0 DO
  >r over wurst-source + c@ r@ c@ xor -rot
    dup >r + $3F and r> r> 8 + LOOP
    drop ;
: bytes2sum ( ud b1 .. b8 -- ud' )
    >r >r >r >r  >r >r >r >r
    r> rngs wurst  r> rngs wurst
    r> rngs wurst  r> rngs wurst
    r> rngs wurst  r> rngs wurst
    r> rngs wurst  r> rngs wurst ;
```

Implementierung Teil 2

Naiv: zu langsam

```
: round ( n -- ) dup 1- swap  8 0 DO
  wurst-state I permut# 64s + 64@ -64swap
  I mix2bytes 2>r bytes2sum
     2r> 64swap nextstate I 64s + 64!
     LOOP 2drop update-state ;
: rounds ( n -- )  message swap
    dup $F and 8 umin 0 ?DO
        I rounds# round
        dup 'round-flags Ith and IF
            swap +entropy swap
        THEN
    LOOP 2drop ;
```

Geschwindigkeit

Naive Implementierung Mit bigForth ca. 1MB/s (Phenom): Sehr langsam (u.a. wegen 32 Bit). Mit Gforth 64 Bit nicht besser.

Unrolling Schleife unrollen und Konstanten vorher berechnen: ca. 6MB/s

C–Code generieren Ebenfalls Schleife unrollen und Konstanten vorher berechnen mit libcc.fs in Gforth: ca. 500MB/s

Geschwindigkeit

Naive Implementierung Mit bigForth ca. 1MB/s (Phenom): Sehr langsam (u.a. wegen 32 Bit). Mit Gforth 64 Bit nicht besser.

Unrolling Schleife unrollen und Konstanten vorher berechnen: ca. 6MB/s

C–Code generieren Ebenfalls Schleife unrollen und Konstanten vorher berechnen mit libcc.fs in Gforth: ca. 500MB/s

Geschwindigkeit

Naive Implementierung Mit bigForth ca. 1MB/s (Phenom): Sehr langsam (u.a. wegen 32 Bit). Mit Gforth 64 Bit nicht besser.

Unrolling Schleife unrollen und Konstanten vorher berechnen: ca. 6MB/s

C–Code generieren Ebenfalls Schleife unrollen und Konstanten vorher berechnen mit libcc.fs in Gforth: ca. 500MB/s

Analyse

ϕ unumkehrbar? Nach n Runden:

Runde 1 Ein Bit im Ausgang geändert tauscht eine Zufallszahl: 64k–Werte zum Ausprobieren in einem 64–Bit–Wort des Status

Runde 2 Alle Status–Wörter sind geändert, jedes hat mindestens eine Zufallszahl getauscht (durchschnittlich 2): 2^{32}Werte zum Ausprobieren *pro Wort!*

Runde 3 Alle Zufallszahlen anders $\longrightarrow$ Statuswort hat keine Ähnlichkeit mehr, 2^{512} Werte zum Ausprobieren

Runde 4 Auch der Akkumulator enthält jetzt komplett andere Daten

Analyse

ϕ unumkehrbar? Nach n Runden:

Runde 1 Ein Bit im Ausgang geändert tauscht eine Zufallszahl: 64k–Werte zum Ausprobieren in einem 64–Bit–Wort des Status

Runde 2 Alle Status–Wörter sind geändert, jedes hat mindestens eine Zufallszahl getauscht (durchschnittlich 2): 2^{32}Werte zum Ausprobieren *pro Wort!*

Runde 3 Alle Zufallszahlen anders $\longrightarrow$ Statuswort hat keine Ähnlichkeit mehr, 2^{512} Werte zum Ausprobieren

Runde 4 Auch der Akkumulator enthält jetzt komplett andere Daten

Analyse

ϕ unumkehrbar? Nach n Runden:

Runde 1 Ein Bit im Ausgang geändert tauscht eine Zufallszahl: 64k–Werte zum Ausprobieren in einem 64–Bit–Wort des Status

Runde 2 Alle Status–Wörter sind geändert, jedes hat mindestens eine Zufallszahl getauscht (durchschnittlich 2): 2^{32}Werte zum Ausprobieren *pro Wort!*

Runde 3 Alle Zufallszahlen anders $\longrightarrow$ Statuswort hat keine Ähnlichkeit mehr, 2^{512} Werte zum Ausprobieren

Runde 4 Auch der Akkumulator enthält jetzt komplett andere Daten

Analyse

ϕ unumkehrbar? Nach n Runden:

Runde 1 Ein Bit im Ausgang geändert tauscht eine Zufallszahl: 64k–Werte zum Ausprobieren in einem 64–Bit–Wort des Status

Runde 2 Alle Status–Wörter sind geändert, jedes hat mindestens eine Zufallszahl getauscht (durchschnittlich 2): 2^{32}Werte zum Ausprobieren *pro Wort!*

Runde 3 Alle Zufallszahlen anders $\longrightarrow$ Statuswort hat keine Ähnlichkeit mehr, 2^{512} Werte zum Ausprobieren

Runde 4 Auch der Akkumulator enthält jetzt komplett andere Daten

Analyse

ϕ unumkehrbar? Nach n Runden:

Runde 1 Ein Bit im Ausgang geändert tauscht eine Zufallszahl: 64k–Werte zum Ausprobieren in einem 64–Bit–Wort des Status

Runde 2 Alle Status–Wörter sind geändert, jedes hat mindestens eine Zufallszahl getauscht (durchschnittlich 2): 2^{32}Werte zum Ausprobieren *pro Wort!*

Runde 3 Alle Zufallszahlen anders $\longrightarrow$ Statuswort hat keine Ähnlichkeit mehr, 2^{512} Werte zum Ausprobieren

Runde 4 Auch der Akkumulator enthält jetzt komplett andere Daten

Wieviele Runden?

Hash Am Ende der Hash–Berechnung brauchen wir also 4 Runden oder mehr. Wir können aber wohl jede Runde einen neuen 512–Bit–Block Nachricht verknüpfen.

Verschlüsselung Pro Nachrichtenblock brauchen wir mindestens 2 Runden, weil sonst durch eine Klartextattacke der komplette interne Zustand erschließbar ist (Klartext z.B. bekannt durch Verwendung eines bekannten Vorspanns)

Wieviele Runden?

Hash Am Ende der Hash–Berechnung brauchen wir also 4 Runden oder mehr. Wir können aber wohl jede Runde einen neuen 512–Bit–Block Nachricht verknüpfen.

Verschlüsselung Pro Nachrichtenblock brauchen wir mindestens 2 Runden, weil sonst durch eine Klartextattacke der komplette interne Zustand erschließbar ist (Klartext z.B. bekannt durch Verwendung eines bekannten Vorspanns)

Anweisungen zum sicheren Gebrauch

Kryptographie ist nur sicher, wenn sie sorgfältig betrieben wird. Die folgenden Anweisungen beschreiben, wie man bekannte Probleme umgeht.

Signatur Signiert wird der Hash–Wert der Datei, indem er mit dem privaten Schlüssel verschüsselt wird.
Empfehlungen:

- Zufälliger Startwert für den Akkumulator
- Bruce Schneier: Öffentlicher Schlüssel als Startwert — IMHO nicht zielführend, weil der ja bekannt ist

Verschlüsselung Zufälliger Startwert ebenfalls ratsam — hilft gegen Known–Plaintext–Attacken. Timestamp kann gegen Replay–Attacken verwendet werden.

Anweisungen zum sicheren Gebrauch

Kryptographie ist nur sicher, wenn sie sorgfältig betrieben wird. Die folgenden Anweisungen beschreiben, wie man bekannte Probleme umgeht.

Signatur Signiert wird der Hash–Wert der Datei, indem er mit dem privaten Schlüssel verschüsselt wird.
Empfehlungen:

- Zufälliger Startwert für den Akkumulator
- Bruce Schneier: Öffentlicher Schlüssel als Startwert — IMHO nicht zielführend, weil der ja bekannt ist

Verschlüsselung Zufälliger Startwert ebenfalls ratsam — hilft gegen Known–Plaintext–Attacken. Timestamp kann gegen Replay–Attacken verwendet werden.

Anweisungen zum sicheren Gebrauch

Kryptographie ist nur sicher, wenn sie sorgfältig betrieben wird. Die folgenden Anweisungen beschreiben, wie man bekannte Probleme umgeht.

Signatur Signiert wird der Hash–Wert der Datei, indem er mit dem privaten Schlüssel verschüsselt wird.
Empfehlungen:

- Zufälliger Startwert für den Akkumulator
- Bruce Schneier: Öffentlicher Schlüssel als Startwert — IMHO nicht zielführend, weil der ja bekannt ist

Verschlüsselung Zufälliger Startwert ebenfalls ratsam — hilft gegen Known–Plaintext–Attacken. Timestamp kann gegen Replay–Attacken verwendet werden.

Anweisungen zum sicheren Gebrauch

Kryptographie ist nur sicher, wenn sie sorgfältig betrieben wird. Die folgenden Anweisungen beschreiben, wie man bekannte Probleme umgeht.

Signatur Signiert wird der Hash–Wert der Datei, indem er mit dem privaten Schlüssel verschüsselt wird.
Empfehlungen:

- Zufälliger Startwert für den Akkumulator
- Bruce Schneier: Öffentlicher Schlüssel als Startwert — IMHO nicht zielführend, weil der ja bekannt ist

Verschlüsselung Zufälliger Startwert ebenfalls ratsam — hilft gegen Known–Plaintext–Attacken. Timestamp kann gegen Replay–Attacken verwendet werden.

Anweisungen zum sicheren Gebrauch

Kryptographie ist nur sicher, wenn sie sorgfältig betrieben wird. Die folgenden Anweisungen beschreiben, wie man bekannte Probleme umgeht.

Signatur Signiert wird der Hash–Wert der Datei, indem er mit dem privaten Schlüssel verschüsselt wird.
Empfehlungen:

- Zufälliger Startwert für den Akkumulator
- Bruce Schneier: Öffentlicher Schlüssel als Startwert — IMHO nicht zielführend, weil der ja bekannt ist

Verschlüsselung Zufälliger Startwert ebenfalls ratsam — hilft gegen Known–Plaintext–Attacken. Timestamp kann gegen Replay–Attacken verwendet werden.