Krypto für net

Reinventing the Internet

Bernd Paysan

Forth–Tagung 2016, Augsburg

Inhalt

3 Jahre nach Snowden

Wie sich die Welt verändert hat

Politiker	Mehr Überwachung, mehr Terrorpanik, mehr Zensur, Cryptowars 3.0 — kannste komplett knicken
Nutzer	wissen jetzt, dass ihre Dick Pics die Dienste interessieren
Software	Viel in Bewegung, sogar WhatsApp ist jetzt verschlüsselt
Protokolle	Sicherheit ist immer noch hinten draufgetackert

3 Jahre nach Snowden

Wie sich die Welt verändert hat

Politiker Mehr Überwachung, mehr Terrorpanik, mehr Zensur, Cryptowars 3.0 — kannste komplett knicken

Nutzer wissen jetzt, dass ihre Dick Pics die Dienste interessieren

Software Viel in Bewegung, sogar WhatsApp ist jetzt verschlüsselt

Protokolle Sicherheit ist immer noch hinten draufgetackert

3 Jahre nach Snowden

Wie sich die Welt verändert hat

Politiker	Mehr Überwachung, mehr Terrorpanik, mehr Zensur, Cryptowars 3.0 — kannste komplett knicken
Nutzer	wissen jetzt, dass ihre Dick Pics die Dienste interessieren
Software	Viel in Bewegung, sogar WhatsApp ist jetzt verschlüsselt
Protokolle	Sicherheit ist immer noch hinten draufgetackert

3 Jahre nach Snowden

Wie sich die Welt verändert hat

Politiker	Mehr Überwachung, mehr Terrorpanik, mehr Zensur, Cryptowars 3.0 — kannste komplett knicken
Nutzer	wissen jetzt, dass ihre Dick Pics die Dienste interessieren
Software	Viel in Bewegung, sogar WhatsApp ist jetzt verschlüsselt
Protokolle	Sicherheit ist immer noch hinten draufgetackert

3 Jahre nach Snowden

Wie sich die Welt verändert hat

Politiker	Mehr Überwachung, mehr Terrorpanik, mehr Zensur, Cryptowars 3.0 — kannste komplett knicken
Nutzer	wissen jetzt, dass ihre Dick Pics die Dienste interessieren
Software	Viel in Bewegung, sogar WhatsApp ist jetzt verschlüsselt
Protokolle	Sicherheit ist immer noch hinten draufgetackert

Die Feinde des Internets

Kriminelle	Malware, DDoS Attacken, Spam, ...
Großfirmen	Geschlossene Plattformen, Zensur, Partner der Dienste, ...
Regierung	Totalüberwachung, Zensur, Cryptoverbot ...
Nutzer	sorglos, uninformiert, nervend (Trolle), ...
Software	aufgebläht, fehlerhaft, unsicher, ...

Die Feinde des Internets

Kriminelle	Malware, DDoS Attacken, Spam, ...
Großfirmen	Geschlossene Plattformen, Zensur, Partner der Dienste, ...
Regierung	Totalüberwachung, Zensur, Cryptoverbot ...
Nutzer	sorglos, uninformiert, nervend (Trolle), ...
Software	aufgebläht, fehlerhaft, unsicher, ...

Die Feinde des Internets

Kriminelle	Malware, DDoS Attacken, Spam, ...
Großfirmen	Geschlossene Plattformen, Zensur, Partner der Dienste, ...
Regierung	Totalüberwachung, Zensur, Cryptoverbot ...
Nutzer	sorglos, uninformiert, nervend (Trolle), ...
Software	aufgebläht, fehlerhaft, unsicher, ...

Die Feinde des Internets

Kriminelle	Malware, DDoS Attacken, Spam, ...
Großfirmen	Geschlossene Plattformen, Zensur, Partner der Dienste, ...
Regierung	Totalüberwachung, Zensur, Cryptoverbot ...
Nutzer	sorglos, uninformiert, nervend (Trolle), ...
Software	aufgebläht, fehlerhaft, unsicher, ...

Die Feinde des Internets

Kriminelle	Malware, DDoS Attacken, Spam, ...
Großfirmen	Geschlossene Plattformen, Zensur, Partner der Dienste, ...
Regierung	Totalüberwachung, Zensur, Cryptoverbot ...
Nutzer	sorglos, uninformiert, nervend (Trolle), ...
Software	aufgebläht, fehlerhaft, unsicher, ...

Wie viele Defekte?

- Dan Geer: „Kauft alle zero–days“ (und fixt sie)
- Annahme: Die Anzahl Bugs ist endlich. Sind sie das?
- Fehlerdichte zwischen 1/100LoC (CMM 1) und <1/10kLoC (Correctness by Construction [3])
- Netzwerkanwendungen und Protokollstacks in Größenordnung 1M–100MLoC
- Wenn wir nicht mit der Bloatware aufhören, wird das nie was
- Deshalb: **Keep it simple!**

Wie viele Defekte?

- Dan Geer: „Kauft alle zero–days“ (und fixt sie)
- Annahme: Die Anzahl Bugs ist endlich. Sind sie das?
- Fehlerdichte zwischen 1/100LoC (CMM 1) und <1/10kLoC (Correctness by Construction [3])
- Netzwerkanwendungen und Protokollstacks in Größenordnung 1M–100MLoC
- Wenn wir nicht mit der Bloatware aufhören, wird das nie was
- Deshalb: **Keep it simple!**

Wie viele Defekte?

- Dan Geer: „Kauft alle zero–days“ (und fixt sie)
- Annahme: Die Anzahl Bugs ist endlich. Sind sie das?
- Fehlerdichte zwischen 1/100LoC (CMM 1) und <1/10kLoC (Correctness by Construction [3])
- Netzwerkanwendungen und Protokollstacks in Größenordnung 1M–100MLoC
- Wenn wir nicht mit der Bloatware aufhören, wird das nie was
- Deshalb: **Keep it simple!**

Wie viele Defekte?

- Dan Geer: „Kauft alle zero–days“ (und fixt sie)
- Annahme: Die Anzahl Bugs ist endlich. Sind sie das?
- Fehlerdichte zwischen 1/100LoC (CMM 1) und <1/10kLoC (Correctness by Construction [3])
- Netzwerkanwendungen und Protokollstacks in Größenordnung 1M–100MLoC
- Wenn wir nicht mit der Bloatware aufhören, wird das nie was
- Deshalb: **Keep it simple!**

Wie viele Defekte?

- Dan Geer: „Kauft alle zero–days“ (und fixt sie)
- Annahme: Die Anzahl Bugs ist endlich. Sind sie das?
- Fehlerdichte zwischen 1/100LoC (CMM 1) und $<$1/10kLoC (Correctness by Construction [3])
- Netzwerkanwendungen und Protokollstacks in Größenordnung 1M–100MLoC
- Wenn wir nicht mit der Bloatware aufhören, wird das nie was
- Deshalb: **Keep it simple!**

Wie viele Defekte?

- Dan Geer: „Kauft alle zero–days“ (und fixt sie)
- Annahme: Die Anzahl Bugs ist endlich. Sind sie das?
- Fehlerdichte zwischen 1/100LoC (CMM 1) und $<$1/10kLoC (Correctness by Construction [3])
- Netzwerkanwendungen und Protokollstacks in Größenordnung 1M–100MLoC
- Wenn wir nicht mit der Bloatware aufhören, wird das nie was
- Deshalb: **Keep it simple!**

Wo sind die Defekte?

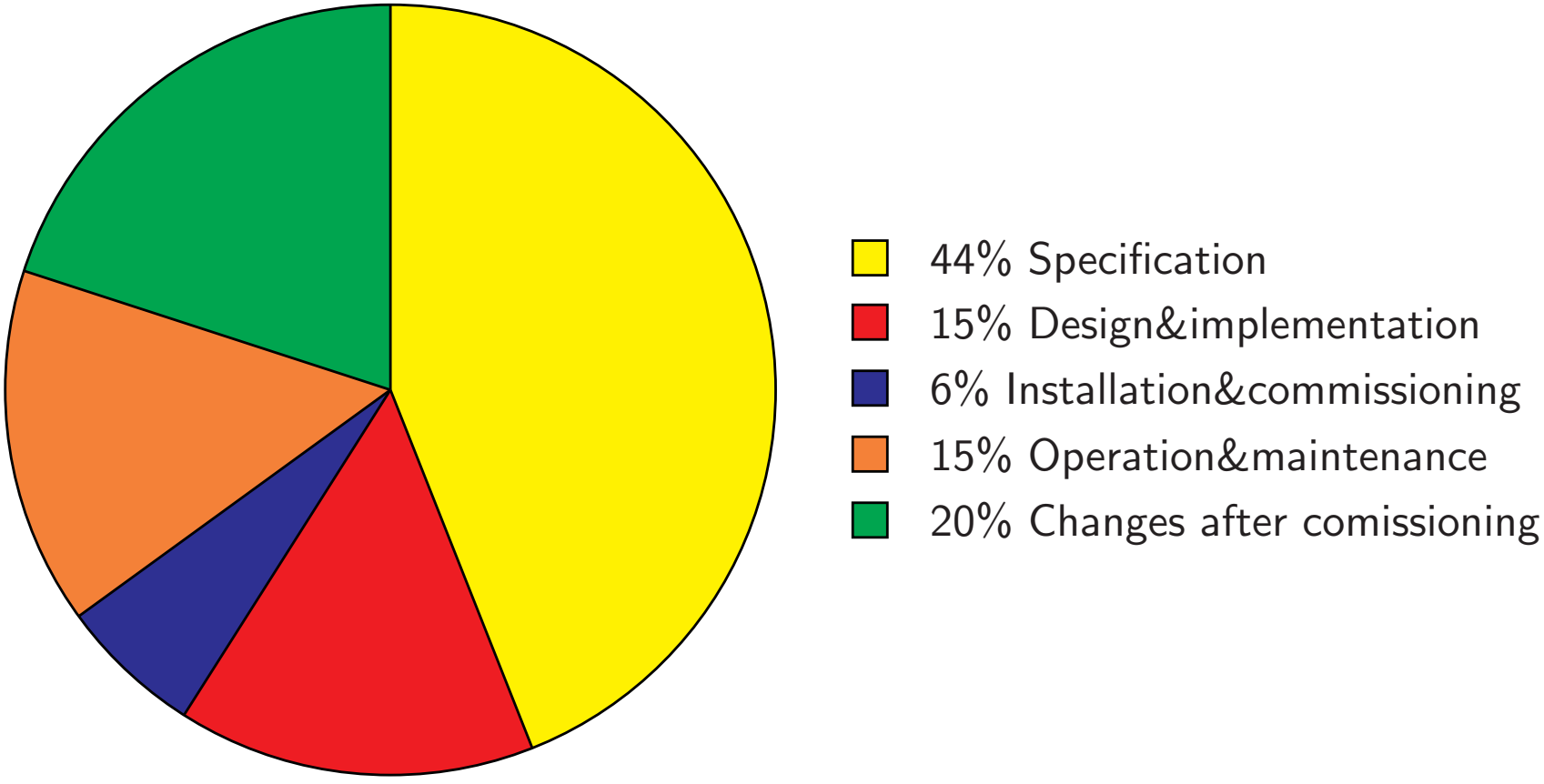
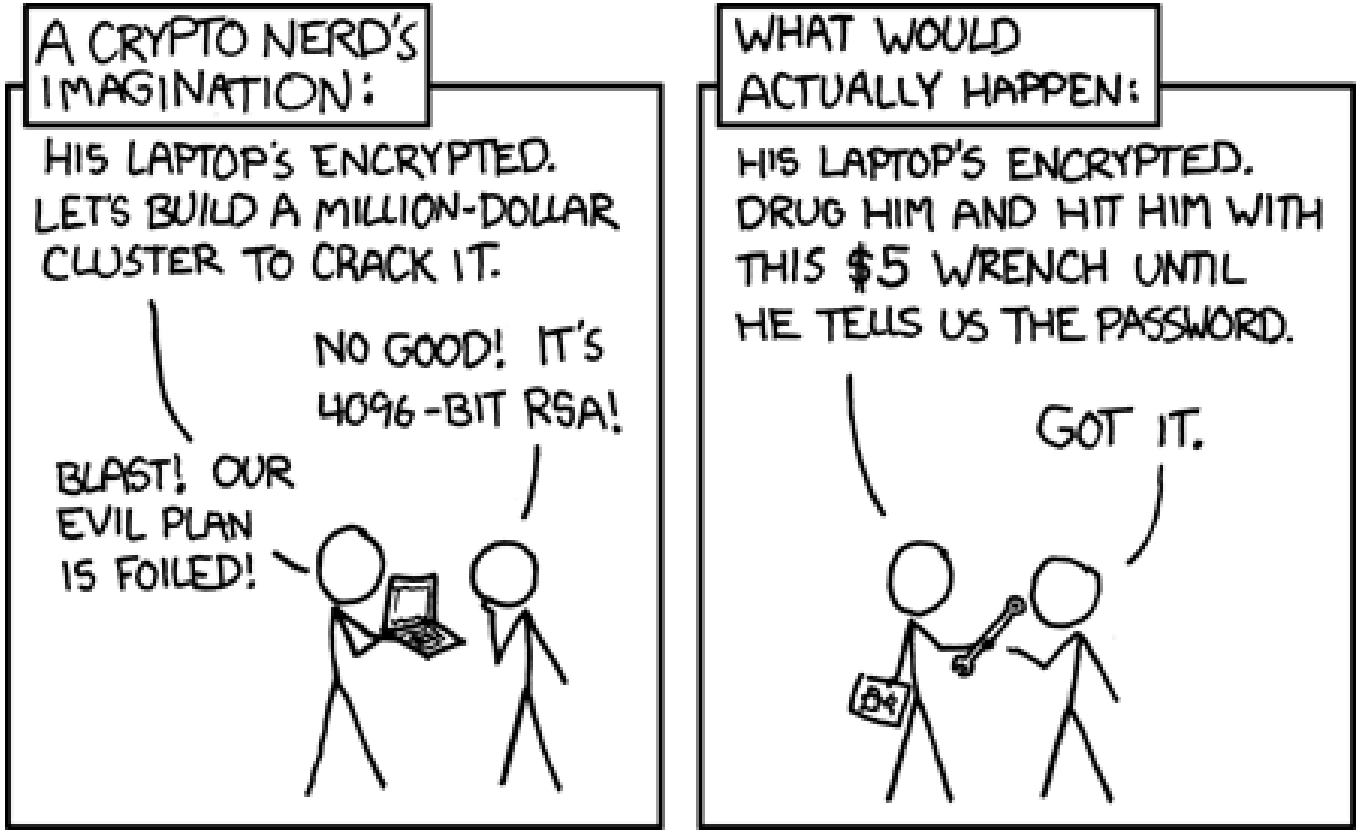


Abbildung: Bugs by phase [2]

Sicherheit: Das schwächste Glied in der Kette



Sicherheit: Das schwächste Glied in der Kette

Verwendungszwecke

Symmetrisch Authentisierte Ver- und Entschlüsselung (AEAD)

Symmetrisch Direkte, blockweise Verschlüsseung

Symmetrisch Hashes

Symmetrisch Zufallsgenerator

Asymmetrisch Schlüsselaustausch

Asymmetrisch Signatur

Verwendungszwecke

Symmetrisch Authentisierte Ver- und Entschlüsselung (AEAD)

Symmetrisch Direkte, blockweise Verschlüsseung

Symmetrisch Hashes

Symmetrisch Zufallsgenerator

Asymmetrisch Schlüsselaustausch

Asymmetrisch Signatur

Verwendungszwecke

Symmetrisch Authentisierte Ver- und Entschlüsselung (AEAD)

Symmetrisch Direkte, blockweise Verschlüsseung

Symmetrisch Hashes

Symmetrisch Zufallsgenerator

Asymmetrisch Schlüsselaustausch

Asymmetrisch Signatur

Verwendungszwecke

Symmetrisch Authentisierte Ver- und Entschlüsselung (AEAD)

Symmetrisch Direkte, blockweise Verschlüsseung

Symmetrisch Hashes

Symmetrisch Zufallsgenerator

Asymmetrisch Schlüsselaustausch

Asymmetrisch Signatur

Verwendungszwecke

Symmetrisch Authentisierte Ver- und Entschlüsselung (AEAD)

Symmetrisch Direkte, blockweise Verschlüsseung

Symmetrisch Hashes

Symmetrisch Zufallsgenerator

Asymmetrisch Schlüsselaustausch

Asymmetrisch Signatur

Verwendungszwecke

Symmetrisch Authentisierte Ver- und Entschlüsselung (AEAD)

Symmetrisch Direkte, blockweise Verschlüsseung

Symmetrisch Hashes

Symmetrisch Zufallsgenerator

Asymmetrisch Schlüsselaustausch

Asymmetrisch Signatur

Asymmetrisch: Ed25519

ECC Elliptische Kurven haben bislang nur eine „generische“ Attacke (baby step/giant step). Damit haben 256 Bit Schlüssellänge eine Sicherheit in der Ordnung von 128 Bits (Speicher+Zeitaufwand zum Knacken)

Deshalb fällt die Wahl auf Ed25519, die Edwards–Form von Curve25519 von Dan Bernstein, die auch ein sicheres, weil deterministisches Signaturverfahren unterstützt
Ich benutze Ed25519 für zwei Operationen:
Diffie–Hellman–Exchange und Signaturen.

Asymmetrisch: Ed25519

ECC Elliptische Kurven haben bislang nur eine „generische“ Attacke (baby step/giant step). Damit haben 256 Bit Schlüssellänge eine Sicherheit in der Ordnung von 128 Bits (Speicher+Zeitaufwand zum Knacken)

Deshalb fällt die Wahl auf Ed25519, die Edwards–Form von Curve25519 von Dan Bernstein, die auch ein sicheres, weil deterministisches Signaturverfahren unterstützt

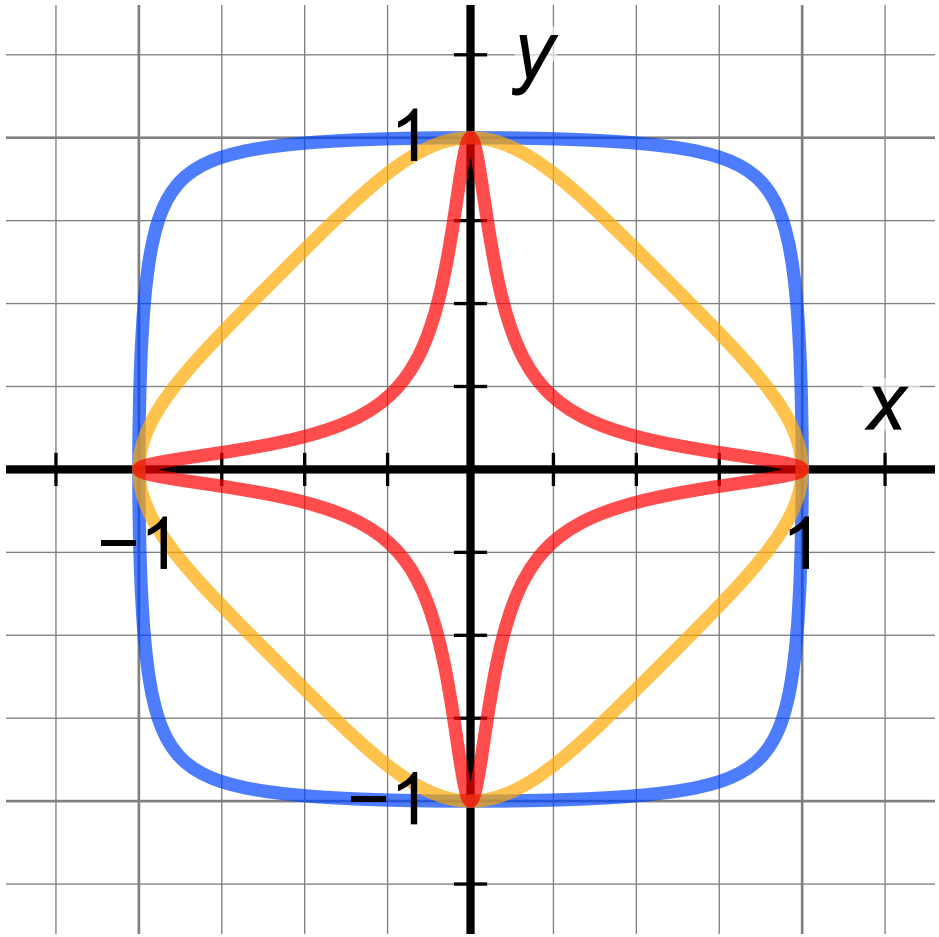
Ich benutze Ed25519 für zwei Operationen:
Diffie–Hellman–Exchange und Signaturen.

Asymmetrisch: Ed25519

ECC Elliptische Kurven haben bislang nur eine „generische“ Attacke (baby step/giant step). Damit haben 256 Bit Schlüssellänge eine Sicherheit in der Ordnung von 128 Bits (Speicher+Zeitaufwand zum Knacken)

Deshalb fällt die Wahl auf Ed25519, die Edwards–Form von Curve25519 von Dan Bernstein, die auch ein sicheres, weil deterministisches Signaturverfahren unterstützt
Ich benutze Ed25519 für zwei Operationen:
Diffie–Hellman–Exchange und Signaturen.

Edwards–Curven



Edwards–Kurve der Gleichung $x^2 + y^2 = 1 - d \cdot x^2 \cdot y^2$ über die reellen Zahlen für $d = 300$ (rot), $d = \sqrt{8}$ (gelb) und $d = -0.9$ (blau)

Edwards–Curven: Winkeladdition

Im Kreis:

$$\sin(\alpha_1 + \alpha_2) = \sin\alpha_1 \cos\alpha_2 + \sin\alpha_2 \cos\alpha_1 = x_1 y_2 + x_2 y_1$$

$$\cos(\alpha_1 + \alpha_2) = \cos\alpha_1 \cos\alpha_2 - \sin\alpha_1 \sin\alpha_2 = y_1 y_2 - x_1 x_2$$

In der Edwards–Kurve:

$$(x_1, y_1) + (x_2, y_2) = \left(\frac{x_1 y_2 + y_1 x_2}{1 + d x_1 x_2 y_1 y_2}, \frac{y_1 y_2 - x_1 x_2}{1 - d x_1 x_2 y_1 y_2} \right)$$

Edwards–Curven: Winkeladdition

Im Kreis:

$$\sin(\alpha_1 + \alpha_2) = \sin\alpha_1 \cos\alpha_2 + \sin\alpha_2 \cos\alpha_1 = x_1 y_2 + x_2 y_1$$

$$\cos(\alpha_1 + \alpha_2) = \cos\alpha_1 \cos\alpha_2 - \sin\alpha_1 \sin\alpha_2 = y_1 y_2 - x_1 x_2$$

In der Edwards–Kurve:

$$(x_1, y_1) + (x_2, y_2) = \left(\frac{x_1 y_2 + y_1 x_2}{1 + d x_1 x_2 y_1 y_2}, \frac{y_1 y_2 - x_1 x_2}{1 - d x_1 x_2 y_1 y_2} \right)$$

Ed25519: Parameter

Punkt–Koordinaten in $\mod (2^{255} - 19)$

Punkte $l =$
$2^{252} + 27742317777372353535851937790883648493$

Basis Ein vorher abgemachter nicht–trivialer Punkt auf der Kurve

Ed25519: Parameter

Punkt–Koordinaten in mod $(2^{255} - 19)$

Punkte $l =$
$2^{252} + 27742317777372353535851937790883648493$

Basis Ein vorher abgemachter nicht–trivialer Punkt auf der Kurve

Ed25519: Parameter

Punkt–Koordinaten in mod $(2^{255} - 19)$

Punkte $l =$
$2^{252} + 27742317777372353535851937790883648493$

Basis Ein vorher abgemachter nicht–trivialer Punkt auf der Kurve

Diffie–Hellman–Exchange

1. Wähle einen zufälligen geheimen Schlüssel $[sk] \in \mathbb{Z}_l$
2. Berechne den Pubkey $pk = [sk] * base$
3. Gemeinsames Geheimnis durch den Skalarproduktcharakter der Kurve:

$$[sk_2] * pk_1 = [sk_2] * [sk_1] * base = [sk_1] * pk_2$$

Diffie–Hellman–Exchange

1. Wähle einen zufälligen geheimen Schlüssel $[sk] \in \mathbb{Z}_l$
2. Berechne den Pubkey $pk = [sk] * base$
3. Gemeinsames Geheimnis durch den Skalarproduktcharakter der Kurve:

$$[sk_2] * pk_1 = [sk_2] * [sk_1] * base = [sk_1] * pk_2$$

Diffie–Hellman–Exchange

1. Wähle einen zufälligen geheimen Schlüssel $[sk] \in \mathbb{Z}_l$
2. Berechne den Pubkey $pk = [sk] * base$
3. Gemeinsames Geheimnis durch den Skalarproduktcharakter der Kurve:

$$[sk_2] * pk_1 = [sk_2] * [sk_1] * base = [sk_1] * pk_2$$

EdDSA–Signatur

1. Generiere eine geheime Pseudozufallszahl $k :=$ mod $(\mathrm{hash512}(\mathrm{absorb}(sk, state)), l)$
2. Bilde den dazugehörigen Punkt auf der Kurve: $r := [k] * base$
3. Berechne $z :=$ mod $(\mathrm{hash512}(\mathrm{absorb}(r||pk, state)), l)$
4. Bilde den zweiten Teil der Signatur: $[s] := [z * sk + k]$
5. Die Signatur ist das Tuple r, s (also 64 Bytes)

EdDSA–Signatur

1. Generiere eine geheime Pseudozufallszahl $k :=$ mod $(\mathrm{hash512}(\mathrm{absorb}(sk, state)), l)$
2. Bilde den dazugehörigen Punkt auf der Kurve: $r := [k] * base$
3. Berechne $z :=$ mod $(\mathrm{hash512}(\mathrm{absorb}(r||pk, state)), l)$
4. Bilde den zweiten Teil der Signatur: $[s] := [z * sk + k]$
5. Die Signatur ist das Tuple r, s (also 64 Bytes)

EdDSA–Signatur

1. Generiere eine geheime Pseudozufallszahl $k :=$ mod $(\mathrm{hash512}(\mathrm{absorb}(sk, state)), l)$
2. Bilde den dazugehörigen Punkt auf der Kurve: $r := [k] * base$
3. Berechne $z :=$ mod $(\mathrm{hash512}(\mathrm{absorb}(r||pk, state)), l)$
4. Bilde den zweiten Teil der Signatur: $[s] := [z * sk + k]$
5. Die Signatur ist das Tuple r, s (also 64 Bytes)

EdDSA–Signatur

1. Generiere eine geheime Pseudozufallszahl $k :=$ mod $(\mathrm{hash512}(\mathrm{absorb}(sk, state)), l)$
2. Bilde den dazugehörigen Punkt auf der Kurve: $r := [k] * base$
3. Berechne $z :=$ mod $(\mathrm{hash512}(\mathrm{absorb}(r||pk, state)), l)$
4. Bilde den zweiten Teil der Signatur: $[s] := [z * sk + k]$
5. Die Signatur ist das Tuple r, s (also 64 Bytes)

EdDSA–Signatur

1. Generiere eine geheime Pseudozufallszahl $k :=$ mod $(\mathrm{hash512}(\mathrm{absorb}(sk, state)), l)$
2. Bilde den dazugehörigen Punkt auf der Kurve: $r := [k] * base$
3. Berechne $z :=$ mod $(\mathrm{hash512}(\mathrm{absorb}(r||pk, state)), l)$
4. Bilde den zweiten Teil der Signatur: $[s] := [z * sk + k]$
5. Die Signatur ist das Tuple r, s (also 64 Bytes)

EdDSA–Signatur verifizieren

1. Berechne $z :=$ mod $(\mathrm{hash512}(\mathrm{absorb}(r||pk, state)), l)$ wie oben
2. Vergleiche

$$r \equiv [s] * base - [z] * pk$$

$$= [z * sk] * base + [k] * base - [z] * [sk] * base = [k] * base$$

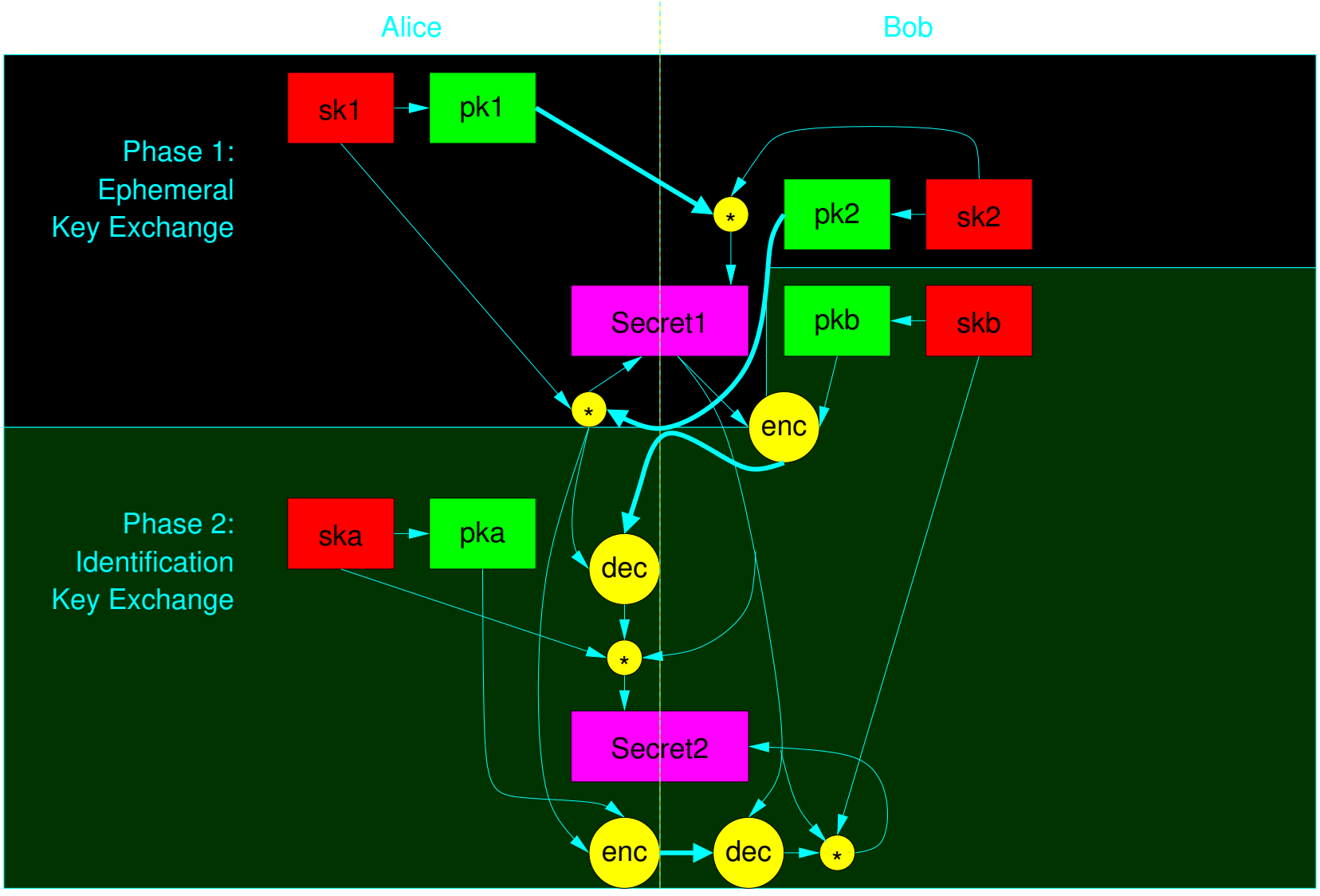
EdDSA–Signatur verifizieren

1. Berechne $z :=$ mod $(\mathrm{hash512}(\mathrm{absorb}(r||pk, state)), l)$ wie oben
2. Vergleiche

$$r \equiv [s] * base - [z] * pk$$

$$= [z * sk] * base + [k] * base - [z] * [sk] * base = [k] * base$$

Ephemeral Key Exchange+Validation



Schlüsselaustausch

Im Fall eines kompromittierten Schlüssels will man nicht den Schlüssel zurückrufen, sondern ihn *ersetzen.* Signieren allein mit dem ersetzten Schlüssel ist nicht ausreichend, weil der ja kompromittiert ist.

- Nur der Erzeuger des geheimen Schlüssels kann ihn ersetzen
- Ein Dieb des Geheimnisses kann das nicht
- Der Rückzug des Schlüssels muss einen authentischen Nachfolgeschlüssel beinhalten
- Bisherige Kommunikationspartner müssen diesem Vorgang direkt vertrauen können
- Lösung: “proof of creation”, d.h. man beweißt, dass man das Geheimnis erzeugt hat.

Schlüsselaustausch

Im Fall eines kompromittierten Schlüssels will man nicht den Schlüssel zurückrufen, sondern ihn *ersetzen.* Signieren allein mit dem ersetzten Schlüssel ist nicht ausreichend, weil der ja kompromittiert ist.

- Nur der Erzeuger des geheimen Schlüssels kann ihn ersetzen
- Ein Dieb des Geheimnisses kann das nicht
- Der Rückzug des Schlüssels muss einen authentischen Nachfolgeschlüssel beinhalten
- Bisherige Kommunikationspartner müssen diesem Vorgang direkt vertrauen können
- Lösung: “proof of creation”, d.h. man beweißt, dass man das Geheimnis erzeugt hat.

Schlüsselaustausch

Im Fall eines kompromittierten Schlüssels will man nicht den Schlüssel zurückrufen, sondern ihn *ersetzen.* Signieren allein mit dem ersetzten Schlüssel ist nicht ausreichend, weil der ja kompromittiert ist.

- Nur der Erzeuger des geheimen Schlüssels kann ihn ersetzen
- Ein Dieb des Geheimnisses kann das nicht
- Der Rückzug des Schlüssels muss einen authentischen Nachfolgeschlüssel beinhalten
- Bisherige Kommunikationspartner müssen diesem Vorgang direkt vertrauen können
- Lösung: “proof of creation”, d.h. man beweißt, dass man das Geheimnis erzeugt hat.

Schlüsselaustausch

Im Fall eines kompromittierten Schlüssels will man nicht den Schlüssel zurückrufen, sondern ihn *ersetzen.* Signieren allein mit dem ersetzten Schlüssel ist nicht ausreichend, weil der ja kompromittiert ist.

- Nur der Erzeuger des geheimen Schlüssels kann ihn ersetzen
- Ein Dieb des Geheimnisses kann das nicht
- Der Rückzug des Schlüssels muss einen authentischen Nachfolgeschlüssel beinhalten
- Bisherige Kommunikationspartner müssen diesem Vorgang direkt vertrauen können
- Lösung: “proof of creation”, d.h. man beweißt, dass man das Geheimnis erzeugt hat.

Schlüsselaustausch

Im Fall eines kompromittierten Schlüssels will man nicht den Schlüssel zurückrufen, sondern ihn *ersetzen.* Signieren allein mit dem ersetzten Schlüssel ist nicht ausreichend, weil der ja kompromittiert ist.

- Nur der Erzeuger des geheimen Schlüssels kann ihn ersetzen
- Ein Dieb des Geheimnisses kann das nicht
- Der Rückzug des Schlüssels muss einen authentischen Nachfolgeschlüssel beinhalten
- Bisherige Kommunikationspartner müssen diesem Vorgang direkt vertrauen können
- Lösung: “proof of creation”, d.h. man beweißt, dass man das Geheimnis erzeugt hat.

Schlüsselaustausch

Im Fall eines kompromittierten Schlüssels will man nicht den Schlüssel zurückrufen, sondern ihn *ersetzen.* Signieren allein mit dem ersetzten Schlüssel ist nicht ausreichend, weil der ja kompromittiert ist.

- Nur der Erzeuger des geheimen Schlüssels kann ihn ersetzen
- Ein Dieb des Geheimnisses kann das nicht
- Der Rückzug des Schlüssels muss einen authentischen Nachfolgeschlüssel beinhalten
- Bisherige Kommunikationspartner müssen diesem Vorgang direkt vertrauen können
- Lösung: “proof of creation”, d.h. man beweißt, dass man das Geheimnis erzeugt hat.

Proof of Creation

- Wähle zwei Zufallszahlen mit je 256 Bits s_1 und s_2
- Erzeuge Pubkeys $p_1 = base * [s_1]$ und $p_2 = base * [s_2]$
- Berechne $[s] = [s_1 * p_2]$ als „Arbeitsgeheimnis“ und $p = base * [s]$, der Pubkey
- Veröffentlihe p und p_1, vernichte s_1 (nicht mehr nötig), behalte s_2 als Offline–Kopie
- Zum Zurückrufen, veröffentliche p_2, der Empfänger kann dann prüfen ob $p_1 * [p_2] \equiv p$.
- Um den Besitz aller Geheimnisse zu beweisen, signiere den neuen Key mit s_2, s, und s_{new}

Proof of Creation

- Wähle zwei Zufallszahlen mit je 256 Bits s_1 und s_2
- Erzeuge Pubkeys $p_1 = base * [s_1]$ und $p_2 = base * [s_2]$
- Berechne $[s] = [s_1 * p_2]$ als „Arbeitsgeheimnis“ und $p = base * [s]$, der Pubkey
- Veröffentlihe p und p_1, vernichte s_1 (nicht mehr nötig), behalte s_2 als Offline–Kopie
- Zum Zurückrufen, veröffentliche p_2, der Empfänger kann dann prüfen ob $p_1 * [p_2] \equiv p$.
- Um den Besitz aller Geheimnisse zu beweisen, signiere den neuen Key mit s_2, s, und s_{new}

Proof of Creation

- Wähle zwei Zufallszahlen mit je 256 Bits s_1 und s_2
- Erzeuge Pubkeys $p_1 = base * [s_1]$ und $p_2 = base * [s_2]$
- Berechne $[s] = [s_1 * p_2]$ als „Arbeitsgeheimnis“ und $p = base * [s]$, der Pubkey
- Veröffentlihe p und p_1, vernichte s_1 (nicht mehr nötig), behalte s_2 als Offline–Kopie
- Zum Zurückrufen, veröffentliche p_2, der Empfänger kann dann prüfen ob $p_1 * [p_2] \equiv p$.
- Um den Besitz aller Geheimnisse zu beweisen, signiere den neuen Key mit s_2, s, und s_{new}

Proof of Creation

- Wähle zwei Zufallszahlen mit je 256 Bits s_1 und s_2
- Erzeuge Pubkeys $p_1 = base * [s_1]$ und $p_2 = base * [s_2]$
- Berechne $[s] = [s_1 * p_2]$ als „Arbeitsgeheimnis“ und $p = base * [s]$, der Pubkey
- Veröffentlihe p und p_1, vernichte s_1 (nicht mehr nötig), behalte s_2 als Offline–Kopie
- Zum Zurückrufen, veröffentliche p_2, der Empfänger kann dann prüfen ob $p_1 * [p_2] \equiv p$.
- Um den Besitz aller Geheimnisse zu beweisen, signiere den neuen Key mit s_2, s, und s_{new}

Proof of Creation

- Wähle zwei Zufallszahlen mit je 256 Bits s_1 und s_2
- Erzeuge Pubkeys $p_1 = base * [s_1]$ und $p_2 = base * [s_2]$
- Berechne $[s] = [s_1 * p_2]$ als „Arbeitsgeheimnis“ und $p = base * [s]$, der Pubkey
- Veröffentlihe p und p_1, vernichte s_1 (nicht mehr nötig), behalte s_2 als Offline–Kopie
- Zum Zurückrufen, veröffentliche p_2, der Empfänger kann dann prüfen ob $p_1 * [p_2] \equiv p$.
- Um den Besitz aller Geheimnisse zu beweisen, signiere den neuen Key mit s_2, s, und s_{new}

Proof of Creation

- Wähle zwei Zufallszahlen mit je 256 Bits s_1 und s_2
- Erzeuge Pubkeys $p_1 = base * [s_1]$ und $p_2 = base * [s_2]$
- Berechne $[s] = [s_1 * p_2]$ als „Arbeitsgeheimnis“ und $p = base * [s]$, der Pubkey
- Veröffentlihe p und p_1, vernichte s_1 (nicht mehr nötig), behalte s_2 als Offline–Kopie
- Zum Zurückrufen, veröffentliche p_2, der Empfänger kann dann prüfen ob $p_1 * [p_2] \equiv p$.
- Um den Besitz aller Geheimnisse zu beweisen, signiere den neuen Key mit s_2, s, und s_{new}

Das PKI–Problem

- Diffie–Hellman bietet keine Sicherheit gegen die Man–in–the–Middle–Attacke, weil der Angreifer beiden Seiten vorgaukelt, er sei der jeweils andere
- Viele ungeeignete Versuche, das über Zertifikate und ähnliches zu lösen
- Beispiel Zertifikat: Ein Zertifikat von z.B. Let's Encrypt bekomme ich, indem mein ACME–Script dort eines beantragt, und Let's Encrypt die Challenge von meinem Web–Server herunterhlädt
- Andere Zertifikate bekommt man, indem eine Verifikations–E–Mail an den Server gesendet wird
- Angriffsvektor: Der Angreifer lenkt alle Zugriffe zum Server auf sich — Ups...
- Lösung: TOFU (Trust on first use) + persönliche Verifikation

Das PKI–Problem

- Diffie–Hellman bietet keine Sicherheit gegen die Man–in–the–Middle–Attacke, weil der Angreifer beiden Seiten vorgaukelt, er sei der jeweils andere
- Viele ungeeignete Versuche, das über Zertifikate und ähnliches zu lösen
- Beispiel Zertifikat: Ein Zertifikat von z.B. Let's Encrypt bekomme ich, indem mein ACME–Script dort eines beantragt, und Let's Encrypt die Challenge von meinem Web–Server herunterhlädt
- Andere Zertifikate bekommt man, indem eine Verifikations–E–Mail an den Server gesendet wird
- Angriffsvektor: Der Angreifer lenkt alle Zugriffe zum Server auf sich — Ups...
- Lösung: TOFU (Trust on first use) + persönliche Verifikation

Das PKI–Problem

- Diffie–Hellman bietet keine Sicherheit gegen die Man–in–the–Middle–Attacke, weil der Angreifer beiden Seiten vorgaukelt, er sei der jeweils andere
- Viele ungeeignete Versuche, das über Zertifikate und ähnliches zu lösen
- Beispiel Zertifikat: Ein Zertifikat von z.B. Let's Encrypt bekomme ich, indem mein ACME–Script dort eines beantragt, und Let's Encrypt die Challenge von meinem Web–Server herunterhlädt
- Andere Zertifikate bekommt man, indem eine Verifikations–E–Mail an den Server gesendet wird
- Angriffsvektor: Der Angreifer lenkt alle Zugriffe zum Server auf sich — Ups...
- Lösung: TOFU (Trust on first use) + persönliche Verifikation

Das PKI–Problem

- Diffie–Hellman bietet keine Sicherheit gegen die Man–in–the–Middle–Attacke, weil der Angreifer beiden Seiten vorgaukelt, er sei der jeweils andere
- Viele ungeeignete Versuche, das über Zertifikate und ähnliches zu lösen
- Beispiel Zertifikat: Ein Zertifikat von z.B. Let's Encrypt bekomme ich, indem mein ACME–Script dort eines beantragt, und Let's Encrypt die Challenge von meinem Web–Server herunterhlädt
- Andere Zertifikate bekommt man, indem eine Verifikations–E–Mail an den Server gesendet wird
- Angriffsvektor: Der Angreifer lenkt alle Zugriffe zum Server auf sich — Ups...
- Lösung: TOFU (Trust on first use) + persönliche Verifikation

Das PKI–Problem

- Diffie–Hellman bietet keine Sicherheit gegen die Man–in–the–Middle–Attacke, weil der Angreifer beiden Seiten vorgaukelt, er sei der jeweils andere
- Viele ungeeignete Versuche, das über Zertifikate und ähnliches zu lösen
- Beispiel Zertifikat: Ein Zertifikat von z.B. Let's Encrypt bekomme ich, indem mein ACME–Script dort eines beantragt, und Let's Encrypt die Challenge von meinem Web–Server herunterhlädt
- Andere Zertifikate bekommt man, indem eine Verifikations–E–Mail an den Server gesendet wird
- Angriffsvektor: Der Angreifer lenkt alle Zugriffe zum Server auf sich — Ups...
- Lösung: TOFU (Trust on first use) + persönliche Verifikation

Das PKI–Problem

- Diffie–Hellman bietet keine Sicherheit gegen die Man–in–the–Middle–Attacke, weil der Angreifer beiden Seiten vorgaukelt, er sei der jeweils andere
- Viele ungeeignete Versuche, das über Zertifikate und ähnliches zu lösen
- Beispiel Zertifikat: Ein Zertifikat von z.B. Let's Encrypt bekomme ich, indem mein ACME–Script dort eines beantragt, und Let's Encrypt die Challenge von meinem Web–Server herunterhlädt
- Andere Zertifikate bekommt man, indem eine Verifikations–E–Mail an den Server gesendet wird
- Angriffsvektor: Der Angreifer lenkt alle Zugriffe zum Server auf sich — Ups...
- Lösung: TOFU (Trust on first use) + persönliche Verifikation

Nochmal Verwendungszwecke

- Authentisierte Ver- und Entschlüsselung (AEAD)
- Direkte, blockweise Verschlüsseung
- Hashes
- Zufallsgenerator

Nochmal Verwendungszwecke

- Authentisierte Ver- und Entschlüsselung (AEAD)
- Direkte, blockweise Verschlüsseung
- Hashes
- Zufallsgenerator

Nochmal Verwendungszwecke

- Authentisierte Ver- und Entschlüsselung (AEAD)
- Direkte, blockweise Verschlüsseung
- Hashes
- Zufallsgenerator

Nochmal Verwendungszwecke

- Authentisierte Ver- und Entschlüsselung (AEAD)
- Direkte, blockweise Verschlüsseung
- Hashes
- Zufallsgenerator

Symmetrische Kryptographie: Keccak

Keccak ist der Gewinner des SHA–3–Wettbewerbs, und wurde ausgewählt wegen:

- Exzellenter Kryptoanalyse im Verlauf des Wettbewerbs
- Keccak ist ein universell einsetzbares Krypto–Primitiv, das neben Hash auch Zufallsgenerator und AEAD–Verschlüsselung in einem Durchgang beherrscht
- Stärke >256 bits, und sehr große Sicherheitsmarge (5 Runden haben schon Stärke 320 Bits, 24 Runden werden gemacht)
- Keccak ist sowohl für Autoritätsgläubige durch die NIST–Absegnung, als auch unabhängig von der NSA, weil der Wettbewerb offen ist. Die Entwickler sind Europäer. Ich verwende Keccak mit $r = 1024$ und Kapazität $c = 576$ wie von den Autoren vorgeschlagen.

Symmetrische Kryptographie: Keccak

Keccak ist der Gewinner des SHA–3–Wettbewerbs, und wurde ausgewählt wegen:

- Exzellenter Kryptoanalyse im Verlauf des Wettbewerbs
- Keccak ist ein universell einsetzbares Krypto–Primitiv, das neben Hash auch Zufallsgenerator und AEAD–Verschlüsselung in einem Durchgang beherrscht
- Stärke >256 bits, und sehr große Sicherheitsmarge (5 Runden haben schon Stärke 320 Bits, 24 Runden werden gemacht)
- Keccak ist sowohl für Autoritätsgläubige durch die NIST–Absegnung, als auch unabhängig von der NSA, weil der Wettbewerb offen ist. Die Entwickler sind Europäer. Ich verwende Keccak mit $r = 1024$ und Kapazität $c = 576$ wie von den Autoren vorgeschlagen.

Symmetrische Kryptographie: Keccak

Keccak ist der Gewinner des SHA–3–Wettbewerbs, und wurde ausgewählt wegen:

- Exzellenter Kryptoanalyse im Verlauf des Wettbewerbs
- Keccak ist ein universell einsetzbares Krypto–Primitiv, das neben Hash auch Zufallsgenerator und AEAD–Verschlüsselung in einem Durchgang beherrscht
- Stärke >256 bits, und sehr große Sicherheitsmarge (5 Runden haben schon Stärke 320 Bits, 24 Runden werden gemacht)
- Keccak ist sowohl für Autoritätsgläubige durch die NIST–Absegnung, als auch unabhängig von der NSA, weil der Wettbewerb offen ist. Die Entwickler sind Europäer. Ich verwende Keccak mit $r = 1024$ und Kapazität $c = 576$ wie von den Autoren vorgeschlagen.

Symmetrische Kryptographie: Keccak

Keccak ist der Gewinner des SHA–3–Wettbewerbs, und wurde ausgewählt wegen:

- Exzellenter Kryptoanalyse im Verlauf des Wettbewerbs
- Keccak ist ein universell einsetzbares Krypto–Primitiv, das neben Hash auch Zufallsgenerator und AEAD–Verschlüsselung in einem Durchgang beherrscht
- Stärke >256 bits, und sehr große Sicherheitsmarge (5 Runden haben schon Stärke 320 Bits, 24 Runden werden gemacht)
- Keccak ist sowohl für Autoritätsgläubige durch die NIST–Absegnung, als auch unabhängig von der NSA, weil der Wettbewerb offen ist. Die Entwickler sind Europäer. Ich verwende Keccak mit $r = 1024$ und Kapazität $c = 576$ wie von den Autoren vorgeschlagen.

Symmetrische Kryptographie: Keccak

Keccak ist der Gewinner des SHA–3–Wettbewerbs, und wurde ausgewählt wegen:

- Exzellenter Kryptoanalyse im Verlauf des Wettbewerbs
- Keccak ist ein universell einsetzbares Krypto–Primitiv, das neben Hash auch Zufallsgenerator und AEAD–Verschlüsselung in einem Durchgang beherrscht
- Stärke >256 bits, und sehr große Sicherheitsmarge (5 Runden haben schon Stärke 320 Bits, 24 Runden werden gemacht)
- Keccak ist sowohl für Autoritätsgläubige durch die NIST–Absegnung, als auch unabhängig von der NSA, weil der Wettbewerb offen ist. Die Entwickler sind Europäer. Ich verwende Keccak mit $r = 1024$ und Kapazität $c = 576$ wie von den Autoren vorgeschlagen.

Keccak Rundenfunktion

θ (Thetha) Lineare Mischoperation

$$p_j \leftarrow a_{0,j} \oplus a_{1,j} \oplus a_{2,j} \oplus a_{3,j} \oplus a_{4,j}$$
$$a_{i,j} \leftarrow a_{i,j} \oplus p_{j-1} \oplus (p_{j+1} \lll 1)$$

ρ (Rho) Wörter des Zustandsvektors rotieren

$a_{i,j} \leftarrow a_{i,j} \lll (w_{i,j} \mod 2^l)$; mit

$w_{0,0} = 0;\ w_{i,j} = \frac{(t+1)(t+2)}{2};\ t \in \{0, \ldots, 23\};$

π (Pi) Wörter des Zustandsvektors permutieren

$a_{j,2i+3j} \leftarrow a_{i,j}$

χ (Chi) Nichtlineare Operation (logisches Und)

$p_{i,j} \leftarrow \neg a_{i,j+1} \wedge a_{i,j+2}$

$a_{i,j} \leftarrow a_{i,j} \oplus p_{i,j}$

ι (Iota) XOR–Verknüpfen mit einer rundenabhängigen Konstanten

$a_{0,0} \leftarrow a_{0,0} \oplus C_r;\ r = 0, 1, \ldots, 11 + 2l$

Keccak Rundenfunktion

θ (Thetha) Lineare Mischoperation

$$p_j \leftarrow a_{0,j} \oplus a_{1,j} \oplus a_{2,j} \oplus a_{3,j} \oplus a_{4,j}$$
$$a_{i,j} \leftarrow a_{i,j} \oplus p_{j-1} \oplus (p_{j+1} \lll 1)$$

ρ (Rho) Wörter des Zustandsvektors rotieren

$$a_{i,j} \leftarrow a_{i,j} \lll (w_{i,j} \mod 2^l); \text{ mit}$$
$$w_{0,0} = 0;\ w_{i,j} = \frac{(t+1)(t+2)}{2};\ t \in \{0, \ldots, 23\};$$

π (Pi) Wörter des Zustandsvektors permutieren

$$a_{j,2i+3j} \leftarrow a_{i,j}$$

χ (Chi) Nichtlineare Operation (logisches Und)

$$p_{i,j} \leftarrow \neg a_{i,j+1} \wedge a_{i,j+2}$$
$$a_{i,j} \leftarrow a_{i,j} \oplus p_{i,j}$$

ι (Iota) XOR–Verknüpfen mit einer rundenabhängigen Konstanten

$$a_{0,0} \leftarrow a_{0,0} \oplus C_r;\ r = 0, 1, \ldots, 11 + 2l$$

Keccak Rundenfunktion

θ (Thetha) Lineare Mischoperation

$$p_j \leftarrow a_{0,j} \oplus a_{1,j} \oplus a_{2,j} \oplus a_{3,j} \oplus a_{4,j}$$
$$a_{i,j} \leftarrow a_{i,j} \oplus p_{j-1} \oplus (p_{j+1} \lll 1)$$

ρ (Rho) Wörter des Zustandsvektors rotieren

$$a_{i,j} \leftarrow a_{i,j} \lll (w_{i,j} \mod 2^l); \text{ mit}$$
$$w_{0,0} = 0;\ w_{i,j} = \frac{(t+1)(t+2)}{2};\ t \in \{0, \ldots, 23\};$$

π (Pi) Wörter des Zustandsvektors permutieren

$$a_{j,2i+3j} \leftarrow a_{i,j}$$

χ (Chi) Nichtlineare Operation (logisches Und)

$$p_{i,j} \leftarrow \neg a_{i,j+1} \wedge a_{i,j+2}$$
$$a_{i,j} \leftarrow a_{i,j} \oplus p_{i,j}$$

ι (Iota) XOR–Verknüpfen mit einer rundenabhängigen Konstanten

$$a_{0,0} \leftarrow a_{0,0} \oplus C_r;\ r = 0, 1, \ldots, 11 + 2l$$

Keccak Rundenfunktion

θ (Thetha) Lineare Mischoperation

$$p_j \leftarrow a_{0,j} \oplus a_{1,j} \oplus a_{2,j} \oplus a_{3,j} \oplus a_{4,j}$$
$$a_{i,j} \leftarrow a_{i,j} \oplus p_{j-1} \oplus (p_{j+1} \lll 1)$$

ρ (Rho) Wörter des Zustandsvektors rotieren

$$a_{i,j} \leftarrow a_{i,j} \lll (w_{i,j} \mod 2^l); \text{ mit}$$
$$w_{0,0} = 0;\ w_{i,j} = \frac{(t+1)(t+2)}{2};\ t \in \{0, \ldots, 23\};$$

π (Pi) Wörter des Zustandsvektors permutieren

$$a_{j,2i+3j} \leftarrow a_{i,j}$$

χ (Chi) Nichtlineare Operation (logisches Und)

$$p_{i,j} \leftarrow \neg a_{i,j+1} \wedge a_{i,j+2}$$
$$a_{i,j} \leftarrow a_{i,j} \oplus p_{i,j}$$

ι (Iota) XOR–Verknüpfen mit einer rundenabhängigen Konstanten

$$a_{0,0} \leftarrow a_{0,0} \oplus C_r;\ r = 0, 1, \ldots, 11 + 2l$$

Keccak Rundenfunktion

θ (Thetha) Lineare Mischoperation

$$p_j \leftarrow a_{0,j} \oplus a_{1,j} \oplus a_{2,j} \oplus a_{3,j} \oplus a_{4,j}$$
$$a_{i,j} \leftarrow a_{i,j} \oplus p_{j-1} \oplus (p_{j+1} \lll 1)$$

ρ (Rho) Wörter des Zustandsvektors rotieren

$$a_{i,j} \leftarrow a_{i,j} \lll (w_{i,j} \mod 2^l); \text{ mit}$$
$$w_{0,0} = 0;\ w_{i,j} = \frac{(t+1)(t+2)}{2};\ t \in \{0, \ldots, 23\};$$

π (Pi) Wörter des Zustandsvektors permutieren

$$a_{j,2i+3j} \leftarrow a_{i,j}$$

χ (Chi) Nichtlineare Operation (logisches Und)

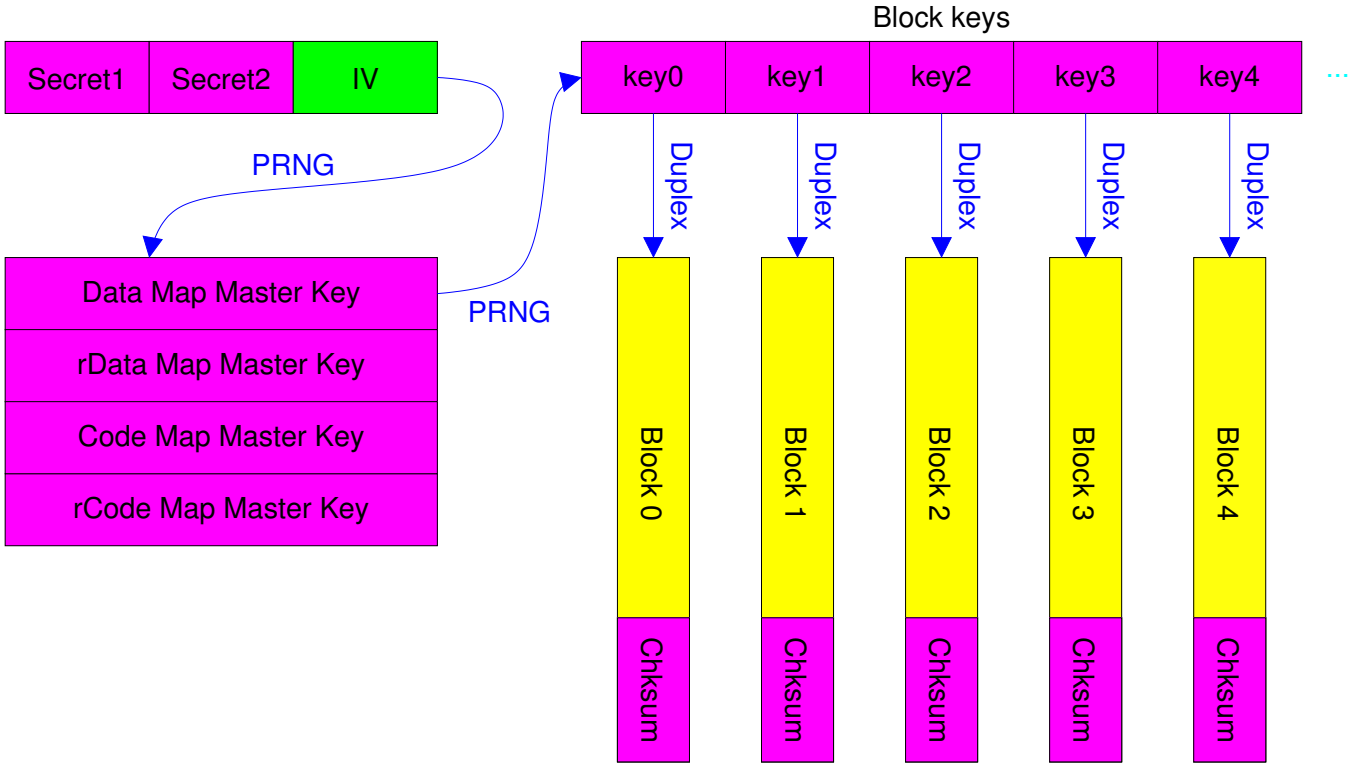
$$p_{i,j} \leftarrow \neg a_{i,j+1} \wedge a_{i,j+2}$$
$$a_{i,j} \leftarrow a_{i,j} \oplus p_{i,j}$$

ι (Iota) XOR–Verknüpfen mit einer rundenabhängigen Konstanten

$$a_{0,0} \leftarrow a_{0,0} \oplus C_r;\ r = 0, 1, \ldots, 11 + 2l$$

Key Usage

All keys are one–time–use only!



Symmetrische Krypto: Threefish

Keccak hat einen Nachteil: Kein ECB–Modus. Braucht man auch nur manchmal:

- Verschlüsselung von Hashes im DHT — um key/value–Paare zu speichern, ohne den Inhalt zu verraten.
- Für net2o–in–net2o tunnels bzw. Onion Routing brauchen wir keine Authentisierung und keinen IV (verlängern das Paket), da reicht ECB (auch AES–ECB).
- Stärke >256 bits, der Tweak macht ECB bei größeren Blöcken sicherer (auch z.B. für Harddisk–Verschlüsselung geeignet).
- Krypto–Primitiv eines SHA–3 Finalists ohne Schwächen, also exzellente Kryptanalyse
- Ich habe einen AEAD–Modus auf SKEIN (der Hash–Funktion von Threefish), die hat aber noch kein ernsthaftes Audit gesehen

Symmetrische Krypto: Threefish

Keccak hat einen Nachteil: Kein ECB–Modus. Braucht man auch nur manchmal:

- Verschlüsselung von Hashes im DHT — um key/value–Paare zu speichern, ohne den Inhalt zu verraten.
- Für net2o–in–net2o tunnels bzw. Onion Routing brauchen wir keine Authentisierung und keinen IV (verlängern das Paket), da reicht ECB (auch AES–ECB).
- Stärke >256 bits, der Tweak macht ECB bei größeren Blöcken sicherer (auch z.B. für Harddisk–Verschlüsselung geeignet).
- Krypto–Primitiv eines SHA–3 Finalists ohne Schwächen, also exzellente Kryptanalyse
- Ich habe einen AEAD–Modus auf SKEIN (der Hash–Funktion von Threefish), die hat aber noch kein ernsthaftes Audit gesehen

Symmetrische Krypto: Threefish

Keccak hat einen Nachteil: Kein ECB–Modus. Braucht man auch nur manchmal:

- Verschlüsselung von Hashes im DHT — um key/value–Paare zu speichern, ohne den Inhalt zu verraten.
- Für net2o–in–net2o tunnels bzw. Onion Routing brauchen wir keine Authentisierung und keinen IV (verlängern das Paket), da reicht ECB (auch AES–ECB).
- Stärke >256 bits, der Tweak macht ECB bei größeren Blöcken sicherer (auch z.B. für Harddisk–Verschlüsselung geeignet).
- Krypto–Primitiv eines SHA–3 Finalists ohne Schwächen, also exzellente Kryptanalyse
- Ich habe einen AEAD–Modus auf SKEIN (der Hash–Funktion von Threefish), die hat aber noch kein ernsthaftes Audit gesehen

Symmetrische Krypto: Threefish

Keccak hat einen Nachteil: Kein ECB–Modus. Braucht man auch nur manchmal:

- Verschlüsselung von Hashes im DHT — um key/value–Paare zu speichern, ohne den Inhalt zu verraten.
- Für net2o–in–net2o tunnels bzw. Onion Routing brauchen wir keine Authentisierung und keinen IV (verlängern das Paket), da reicht ECB (auch AES–ECB).
- Stärke >256 bits, der Tweak macht ECB bei größeren Blöcken sicherer (auch z.B. für Harddisk–Verschlüsselung geeignet).
- Krypto–Primitiv eines SHA–3 Finalists ohne Schwächen, also exzellente Kryptanalyse
- Ich habe einen AEAD–Modus auf SKEIN (der Hash–Funktion von Threefish), die hat aber noch kein ernsthaftes Audit gesehen

Symmetrische Krypto: Threefish

Keccak hat einen Nachteil: Kein ECB–Modus. Braucht man auch nur manchmal:

- Verschlüsselung von Hashes im DHT — um key/value–Paare zu speichern, ohne den Inhalt zu verraten.
- Für net2o–in–net2o tunnels bzw. Onion Routing brauchen wir keine Authentisierung und keinen IV (verlängern das Paket), da reicht ECB (auch AES–ECB).
- Stärke >256 bits, der Tweak macht ECB bei größeren Blöcken sicherer (auch z.B. für Harddisk–Verschlüsselung geeignet).
- Krypto–Primitiv eines SHA–3 Finalists ohne Schwächen, also exzellente Kryptanalyse
- Ich habe einen AEAD–Modus auf SKEIN (der Hash–Funktion von Threefish), die hat aber noch kein ernsthaftes Audit gesehen

Symmetrische Krypto: Threefish

Keccak hat einen Nachteil: Kein ECB–Modus. Braucht man auch nur manchmal:

- Verschlüsselung von Hashes im DHT — um key/value–Paare zu speichern, ohne den Inhalt zu verraten.
- Für net2o–in–net2o tunnels bzw. Onion Routing brauchen wir keine Authentisierung und keinen IV (verlängern das Paket), da reicht ECB (auch AES–ECB).
- Stärke >256 bits, der Tweak macht ECB bei größeren Blöcken sicherer (auch z.B. für Harddisk–Verschlüsselung geeignet).
- Krypto–Primitiv eines SHA–3 Finalists ohne Schwächen, also exzellente Kryptanalyse
- Ich habe einen AEAD–Modus auf SKEIN (der Hash–Funktion von Threefish), die hat aber noch kein ernsthaftes Audit gesehen

Einsatzbeispiel: net2o Vaults

Vaults sind verschlüsselte Dateien, die von mehreren Empfängern geöffnet werden können, als solche Unbefugten aber keine Medataten verraten. Vaults bestehen aus vier Teilen

1. Einem (ephemeral) Pubkey für einen Diffie–Hellman–Exchange mit dem geheimen Schlüssel des Empfängers
2. Einem Block Session–Keys, jeder verschlüsselt im AEAD–Modus mit dem per DHE erzeugten Geheimnis und Initialisierungsvektor (alle Session Keys sind identisch, aber unterschiedlich verschlüsselt)
3. Die eigentliche Datei, verschlüsselt mit dem Session Key
4. Einer ebenfalls mit dem Session–Key verschlüsselten digitalen Signatur der Datei selbst

Einsatzbeispiel: net2o Vaults

Vaults sind verschlüsselte Dateien, die von mehreren Empfängern geöffnet werden können, als solche Unbefugten aber keine Medataten verraten. Vaults bestehen aus vier Teilen

1. Einem (ephemeral) Pubkey für einen Diffie–Hellman–Exchange mit dem geheimen Schlüssel des Empfängers
2. Einem Block Session–Keys, jeder verschlüsselt im AEAD–Modus mit dem per DHE erzeugten Geheimnis und Initialisierungsvektor (alle Session Keys sind identisch, aber unterschiedlich verschlüsselt)
3. Die eigentliche Datei, verschlüsselt mit dem Session Key
4. Einer ebenfalls mit dem Session–Key verschlüsselten digitalen Signatur der Datei selbst

Einsatzbeispiel: net2o Vaults

Vaults sind verschlüsselte Dateien, die von mehreren Empfängern geöffnet werden können, als solche Unbefugten aber keine Medataten verraten. Vaults bestehen aus vier Teilen

1. Einem (ephemeral) Pubkey für einen Diffie–Hellman–Exchange mit dem geheimen Schlüssel des Empfängers
2. Einem Block Session–Keys, jeder verschlüsselt im AEAD–Modus mit dem per DHE erzeugten Geheimnis und Initialisierungsvektor (alle Session Keys sind identisch, aber unterschiedlich verschlüsselt)
3. Die eigentliche Datei, verschlüsselt mit dem Session Key
4. Einer ebenfalls mit dem Session–Key verschlüsselten digitalen Signatur der Datei selbst

Einsatzbeispiel: net2o Vaults

Vaults sind verschlüsselte Dateien, die von mehreren Empfängern geöffnet werden können, als solche Unbefugten aber keine Medataten verraten. Vaults bestehen aus vier Teilen

1. Einem (ephemeral) Pubkey für einen Diffie–Hellman–Exchange mit dem geheimen Schlüssel des Empfängers
2. Einem Block Session–Keys, jeder verschlüsselt im AEAD–Modus mit dem per DHE erzeugten Geheimnis und Initialisierungsvektor (alle Session Keys sind identisch, aber unterschiedlich verschlüsselt)
3. Die eigentliche Datei, verschlüsselt mit dem Session Key
4. Einer ebenfalls mit dem Session–Key verschlüsselten digitalen Signatur der Datei selbst

Einsatzbeispiel: net2o Vaults

Vaults sind verschlüsselte Dateien, die von mehreren Empfängern geöffnet werden können, als solche Unbefugten aber keine Medataten verraten. Vaults bestehen aus vier Teilen

1. Einem (ephemeral) Pubkey für einen Diffie–Hellman–Exchange mit dem geheimen Schlüssel des Empfängers
2. Einem Block Session–Keys, jeder verschlüsselt im AEAD–Modus mit dem per DHE erzeugten Geheimnis und Initialisierungsvektor (alle Session Keys sind identisch, aber unterschiedlich verschlüsselt)
3. Die eigentliche Datei, verschlüsselt mit dem Session Key
4. Einer ebenfalls mit dem Session–Key verschlüsselten digitalen Signatur der Datei selbst

For Further Reading I

Bernd Paysan

net2o source repository and wiki

https://fossil.net2o.de/net2o

Health & Safety Executive HSE – UK

Out of control, 2nd edition 2003

http://www.hse.gov.uk/pubns/priced/hsg238.pdf

Martin Croxford and Dr. Roderick Chapman

Correctness by Construction: A Manifesto for High-Integrity Software

http://www.crosstalkonline.org/storage/issue-archives/2005/200512/200512-Croxford.pdf