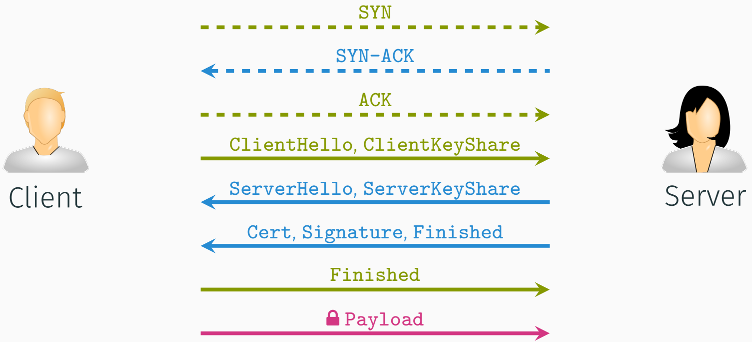
Bloom Filter Encryption and Applications to Efficient Forward-Secret 0-RTT Key Exchange

David Derler[‡], Tibor Jager[||], Daniel Slamanig[§], Christoph Striecks[§]

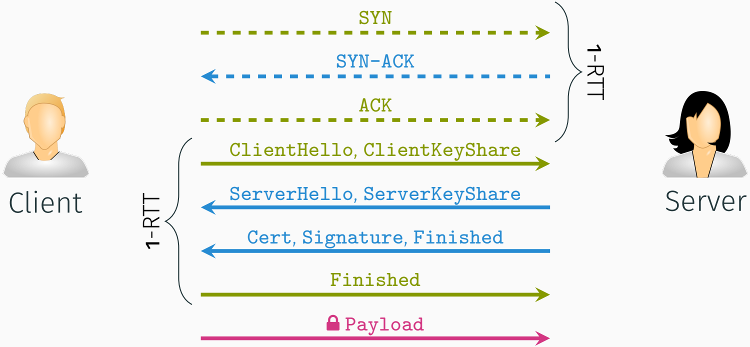
May 3, 2018—Eurocrypt 2018, Tel Aviv, Israel

‡ TU Graz || Paderborn University § AIT Austrian Institute of Technology

Key Establishment with TLS



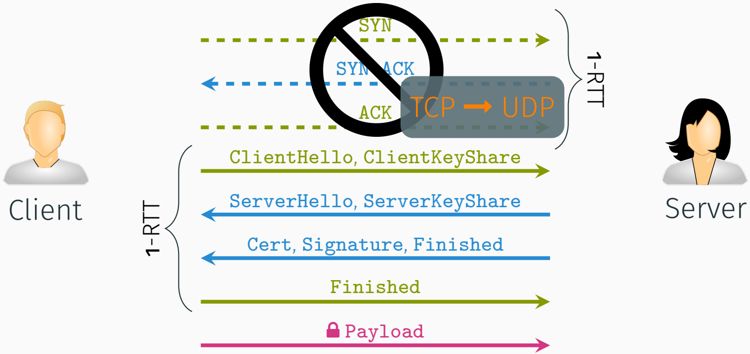
Key Establishment with TLS



» 2-RTTs before first payload message

? Is this necessary

Key Establishment with TLS

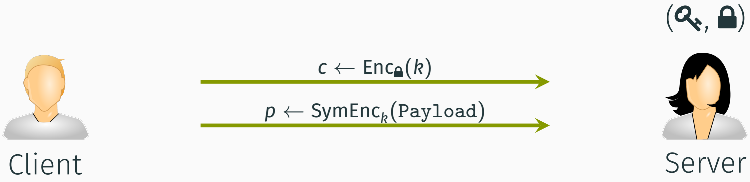


» 2-RTTs before first payload message

? Is this necessary

Send cryptographically protected payload in first message (0-RTT KE)?

Trivial Protocol



Major deficiencies:

- No forward secrecy
- Vulnerable to replay attacks

Existing Approaches

0-RTT in TLS1.3/QUIC

- First session 1-RTT, session resumption 0-RTT
- ✔ Replay protection
- ? Forward secrecy for most transmitted data

Existing Approaches

0-RTT in TLS1.3/QUIC

- First session 1-RTT, session resumption 0-RTT
- ✔ Replay protection
- ? Forward secrecy for most transmitted data

Full forward secrecy, replay protection, and 0-RTT?

- A priori not even clear if possible
- Günther, Hale, Jager, and Lauer at EUROCRYPT'17
- » Using puncturable encryption (Green, Miers at S&P 2015)

Puncturable Encryption

Conventional encryption scheme:

- $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$

+ Additional algorithm 🔑$' \leftarrow \mathsf{Punc}($🔑$, C)$

Properties

- 🔑$'$ no longer useful to decrypt C
- 🔑$'$ still useful to decrypt other ciphertexts
- Repeated puncturing possible

Puncturable Encryption

Conventional encryption scheme:

- (**KeyGen**, **Enc**, **Dec**)

+ Additional algorithm 🔑′ $\leftarrow$ **Punc**(🔑, C)

Properties

- 🔑′ no longer useful to decrypt C
- 🔑′ still useful to decrypt other ciphertexts
- Repeated puncturing possible

fs 0-RTT KE via puncturable encryption

- Client encrypts message under public key 🔒
- Server decrypts using secret key 🔑′
- Server punctures 🔑′ on C

Our Approach

Downsides of existing approaches

- Puncturing and/or decryption expensive

  (experiments by authors of [GHJL17]: 30s - several minutes)

Our Approach

Downsides of existing approaches

- Puncturing and/or decryption expensive

  (experiments by authors of [GHJL17]: 30s - several minutes)

Observation

- Can accept somewhat larger (secret) keys
- Can accept non-negligible correctness error
- For example, 1 in 1000 sessions fail

» Can fall back to 1-RTT in this case

Bloom Filters

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1 m

- Initial state $T := 0^m$
- k universal hash functions $(H_j)_{j \in [k]}$
- $H_j : \mathcal{U} \to [m]$
- Throughout this talk, let $k = 3$

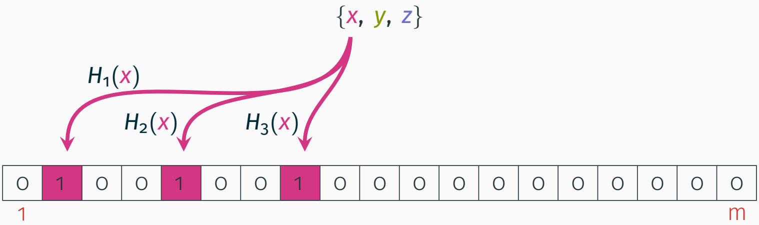
Bloom Filters

$\{x, y, z\}$

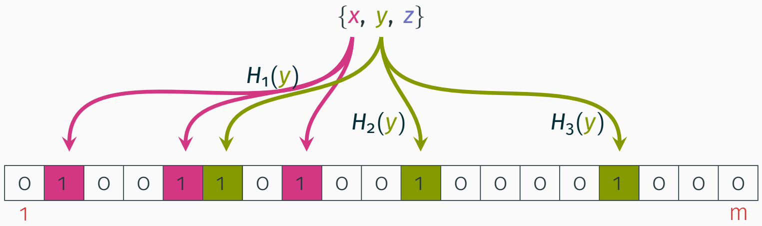
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1																		m

- Initial state $T := 0^m$
- k universal hash functions $(H_j)_{j \in [k]}$
- $H_j : \mathcal{U} \to [m]$
- Throughout this talk, let $k = 3$

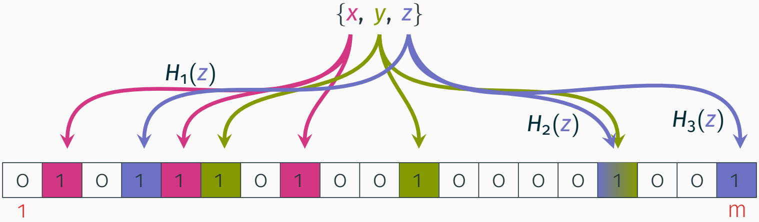
Bloom Filters



Bloom Filters



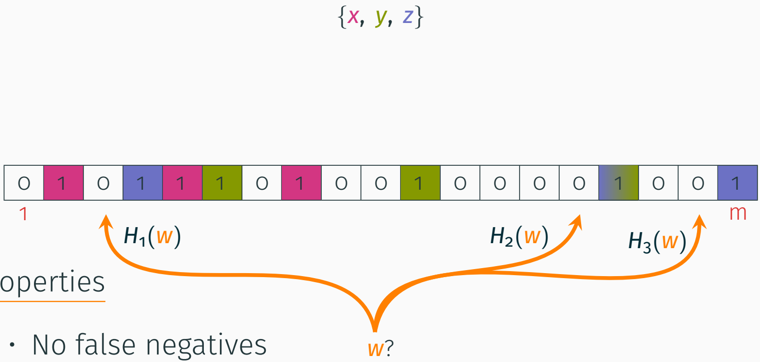
Bloom Filters



Properties

- No false negatives

Bloom Filters

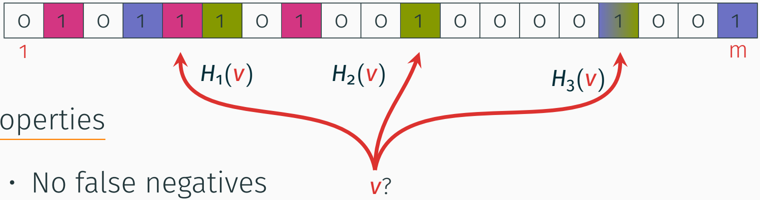


Properties

- No false negatives

Bloom Filters

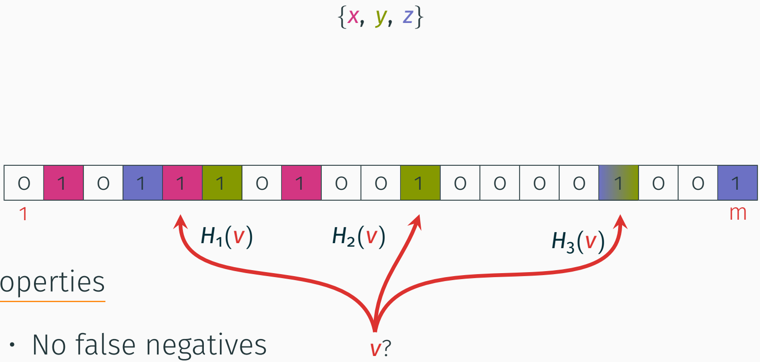
$\{x, y, z\}$



Properties

- No false negatives
- **False positives possible**

Bloom Filters



Properties

- No false negatives
- **False positives possible**
- Probability determined by k, m, and # inserted elements

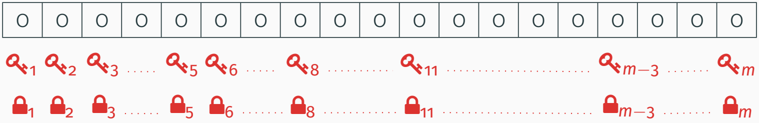
Bloom Filter Encryption

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

KeyGen

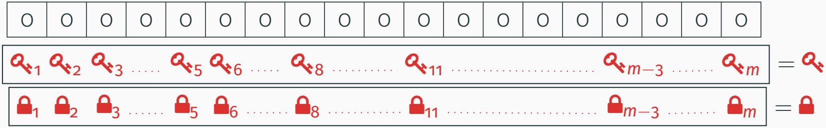
- Set up BF

Bloom Filter Encryption



KeyGen

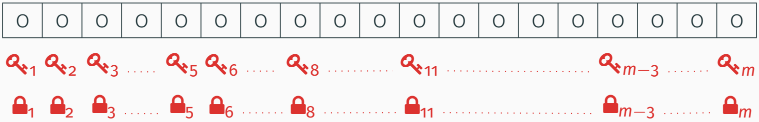
- Set up BF
- Associate key pair to each bit



KeyGen

- Set up BF
- Associate key pair to each bit
- Compose BFE key pair (,)

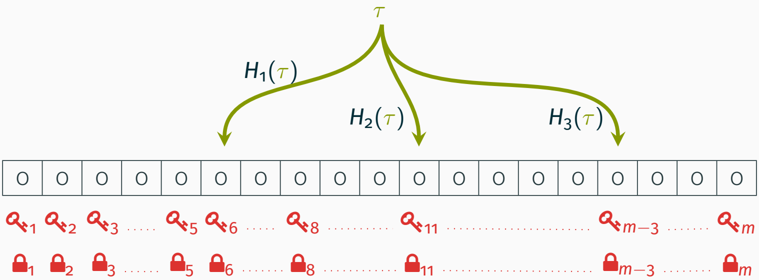
Bloom Filter Encryption



Encrypt message M

- Randomly choose tag τ

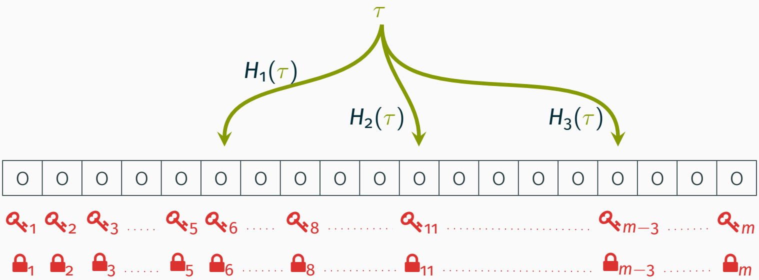
Bloom Filter Encryption



Encrypt message M

- Randomly choose tag τ
- Determine indexes from τ

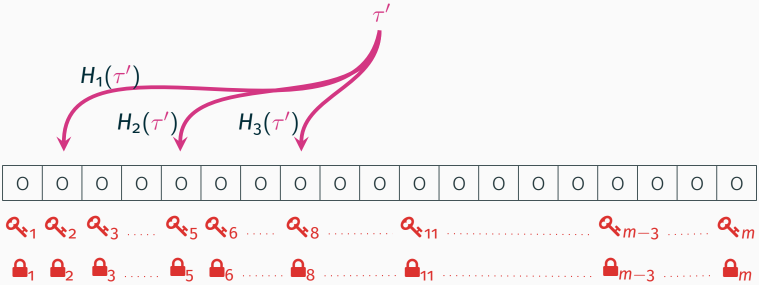
Bloom Filter Encryption



Encrypt message M

- Randomly choose tag τ
- Determine indexes from τ
- $C_\tau \leftarrow \mathsf{Enc}_{🔒_6 \vee 🔒_{11} \vee 🔒_{m-3}}(M)$

Bloom Filter Encryption

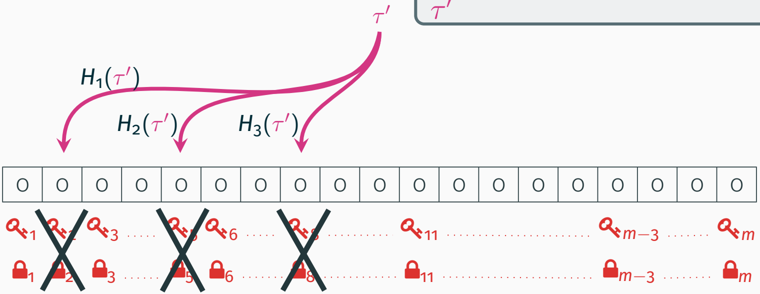


Puncture ciphertext $C_{\tau'}$

- Determine BF indexes from τ'

Bloom Filter Encryption

ℹ Secret key no longer useful to decrypt $C_{\tau'}$ with associated tag τ'

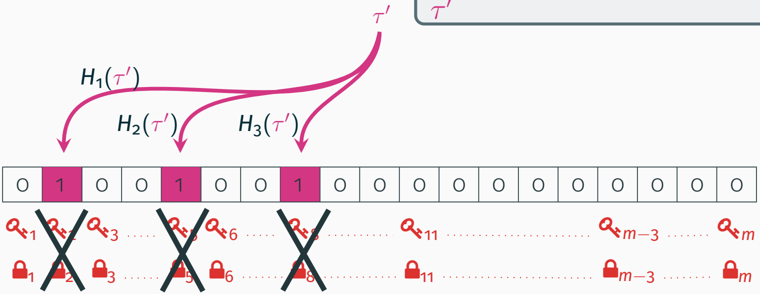


Puncture ciphertext $C_{\tau'}$

- Determine BF indexes from τ'
- Delete associated keys

Bloom Filter Encryption

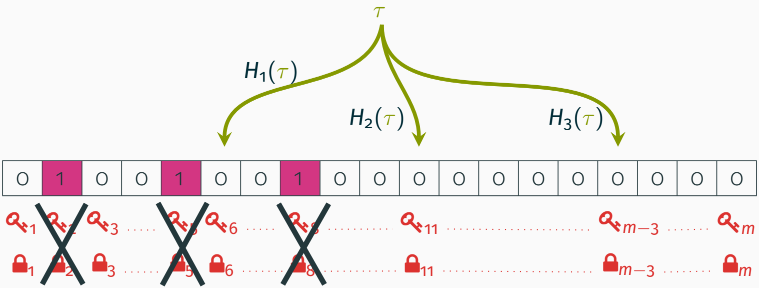
Secret key no longer useful to decrypt $C_{\tau'}$ with associated tag τ'



Puncture ciphertext $C_{\tau'}$

- Determine BF indexes from τ'
- Delete associated keys
- Update BF state

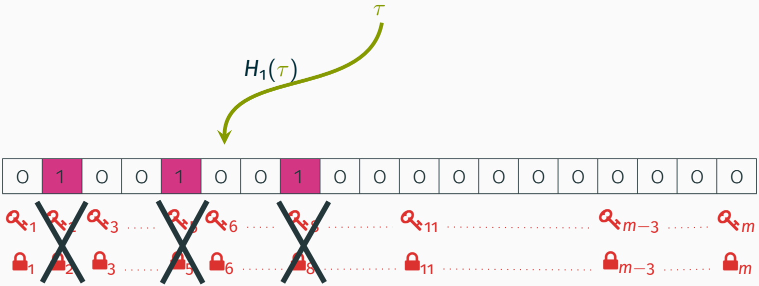
Bloom Filter Encryption



Decrypt ciphertext C_τ

- Determine BF indexes from τ

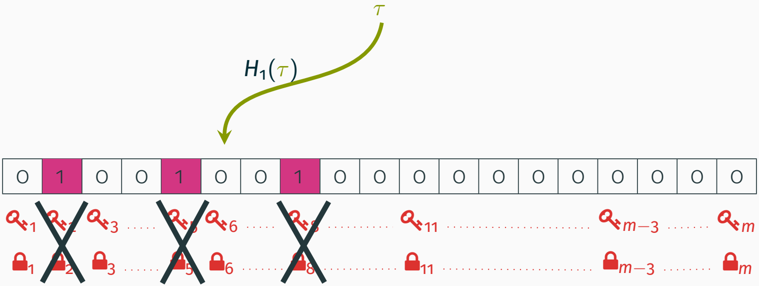
Bloom Filter Encryption



Decrypt ciphertext C_τ

- Determine BF indexes from τ
- Let i lowest index w. $BF[i] = 0$

Bloom Filter Encryption



Decrypt ciphertext C_τ

- Determine BF indexes from τ
- Let i lowest index w. $BF[i] = 0$
- $M \leftarrow \mathsf{Dec}_{sk_6}(C_\tau)$

Example BF Parameters

We let

- Maximum # of elements in BF: 2^{20}

$\approx$ 2^{12} puncturings/day for full year

- False positive probability: 10^{-3}

Then we get

- BF size $m = {}^{n \ln p}/_{(\ln 2)^2} \approx 2MB$
- # hash functions $k = \lceil {}^{m}/_{n} \ln 2 \rceil = 10$

Instantiations

Three instantiations with different trade-offs

» Identity-based encryption (IBE)

» Attribute-based encryption (ABE)

NEW Identity-based broadcast encryption (IBBE)[1]

[1]Construction by Kai Gellert in extended version (ePrint 2018/199)

Instantiations

Three instantiations with different trade-offs

» Identity-based encryption (IBE)

» Attribute-based encryption (ABE)

NEW Identity-based broadcast encryption (IBBE)[1]

Construction	|🔒|	|🔑|	$\lvert C \rvert$	Dec	Punc
IBE [Crypto'01]	$O(1)$	$O(m)$	$O(k)$	$O(k)$	$O(k)$
ABE [CT-RSA'13, AC'15]	$O(m)$	$O(m^2)$	$O(1)$	$O(k)$	$O(k)$
IBBE [AC'07]	$O(k)$	$O(m)$	$O(1)$	$O(k)$	$O(k)$

[1] Construction by Kai Gellert in extended version (ePrint 2018/199)

Instantiations (IBE)

Based on Boneh-Franklin (BF) IBE

- Constant size public key (400 bit at 120 bit security)
- Secret key: include one IBE-🔑 per bit of BF (=identity)

Instantiations (IBE)

Based on Boneh-Franklin (BF) IBE

- Constant size public key (400 bit at 120 bit security)
- Secret key: include one IBE-🔑 per bit of BF (=identity)
- Ciphertext
  - » k BF ciphertexts w. shared rand.
  - » Use hashed variant to save space
  - » Size $\mathcal{O}(k)$
  - $\approx$ 3000 bit (120 bit security, parameters from before)

Instantiations (IBE)

Based on Boneh-Franklin (BF) IBE

- Constant size public key (400 bit at 120 bit security)
- Secret key: include one IBE-🔑 per bit of BF (=identity)
- Ciphertext
  - » k BF ciphertexts w. shared rand.
  - » Use hashed variant to save space
  - » Size $\mathcal{O}(k)$
  - $\approx$ 3000 bit (120 bit security, parameters from before)
- Secret key size $\approx$700MB (parameters from before)

Instantiations (CCA Security)

Fujisaki-Okamoto (FO) transformation

- Use RO to simulate decryption oracle
- Requires perfect correctness

  (Recently negl. correctness error) [Hofheinz et al., TCC'17]

Instantiations (CCA Security)

Fujisaki-Okamoto (FO) transformation

- Use RO to simulate decryption oracle
- Requires perfect correctness

(Recently negl. correctness error) [Hofheinz et al., TCC'17]

BFE has non-negl. correctness error

- Formalize additional properties

» Extended correctness
  - No false-negatives
  - Original keys have perfect correctness
  - Semi correctness of punctured keys

» Publicly-checkable puncturing

✔ Perfect simulation of decryption oracle

Instantiations (CCA Security)

Fujisaki-Okamoto (FO) transformation

- Use RO to simulate decryption oracle
- Requires perfect correctness

  (Recently negl. correctness error) [Hofheinz et al., TCC'17]

BFE has non-negl. correctness error

- Formalize additional properties

» Extended correctness
  - No false-negatives
  - Original keys have perfect correctness
  - Semi correctness of punctured keys

» Publicly-checkable puncturing

✔ Perfect simulation of decryption oracle

Works generically for all our approaches!

Instantiations contd’

Extensions

- Time-based BFE (TBBFE)
- Enable multiple time intervals
- Similar approach as [GM S&P’15, GHJL EC’17]

Use hierarchical identity-based encryption (HIBE) scheme

- Tree of identities

» Upper part represent time intervals

» Lower part represent the bits of BF

(as in BFE)

Comparison of TB-BFEs

Scheme	**Dec** (online)	**PuncCtx** (online)	**PuncInt** (offline)
2^w time slots			
GM [S&P'15]	$O(p)$	$O(1)$	$O(w^2)$
GHJL [EC'17]	$O(\lambda^2)$	$O(\lambda^2)$	$O(w^2)$
Ours	$O(k)$	$O(k)$	$O(w^2 + m)$

With m size of BF, k # hash functions (e.g., $k = 10$), $\lambda \geq 120$, p number of puncturings already performed

Conclusions

Existing approaches

- Most critical ops expensive (puncturing & decryption)
  - ❗ Authors of [GHJL17] report 30s to minutes

Conclusions

Existing approaches

- Most critical ops expensive (puncturing & decryption)
  - ! Authors of [GHJL17] report 30s to minutes

Our approach

- ✔ Offload expensive ops to less critical phases (key generation, resp. switch of time interval for TB)
- ✔ Very efficient decryption
- ✔ Only deletions & hash evaluations upon puncture
- ✔ Conjectured dec. & punc. times in order of milliseconds
- ✔ Applications of BFE beyond 0-RTT KE?

Thank you!

Full version: `https://eprint.iacr.org/2018/199`