Breaking and Fixing Content-Defined Chunking

Kien Tuong Truong, Simon-Philipp Merz, Matteo Scarlata, Felix Günther and Kenny Paterson

Real World Cryptography 2025

Chunking?!?

Chunking?!?

Chunking?!?

Chunking?!?

Flickr user Pete

Chunking?!?

Chunking?!?

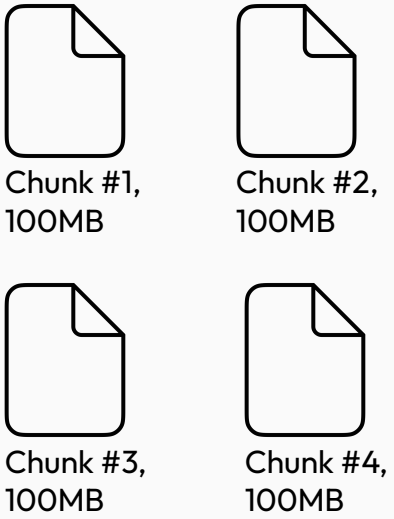
Big_Buck_Bunny_4K.webm, 2.76 GB

Chunking?!?

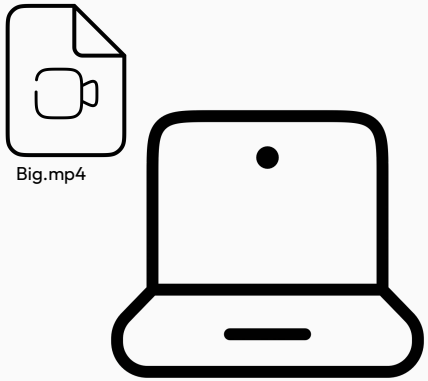
Big_Buck_Bunny_4K.webm, 2.76 GB

Chunking?!?

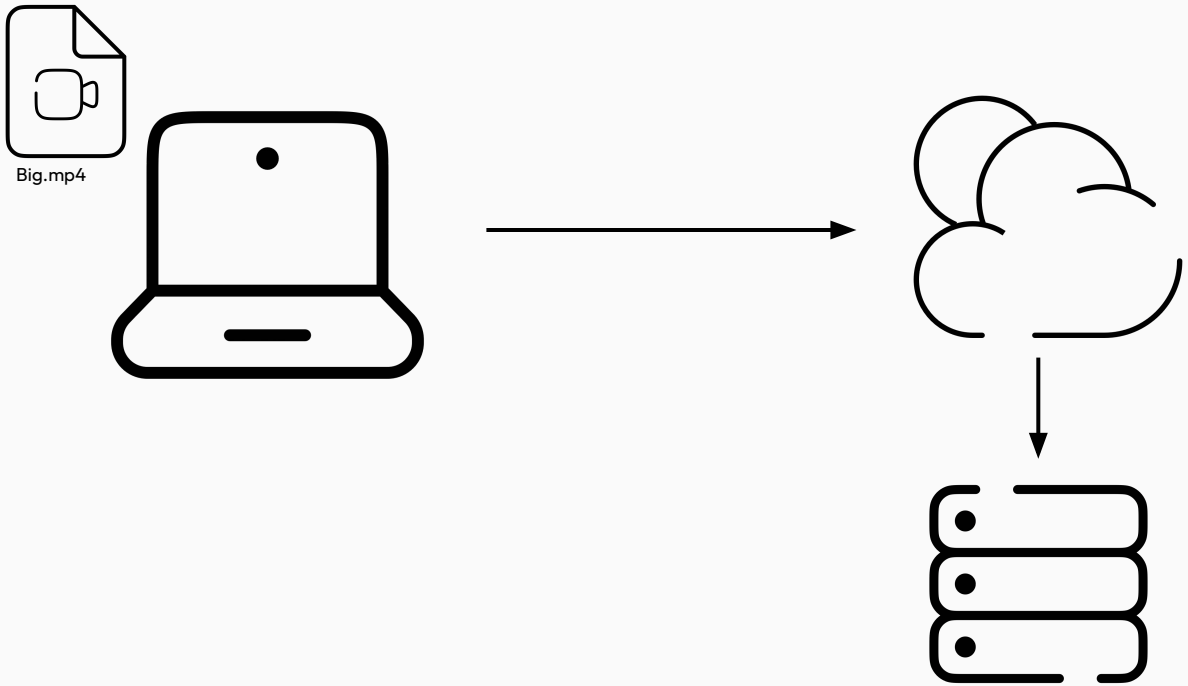
Big_Buck_Bunny_4K.webm, 2.76 GB



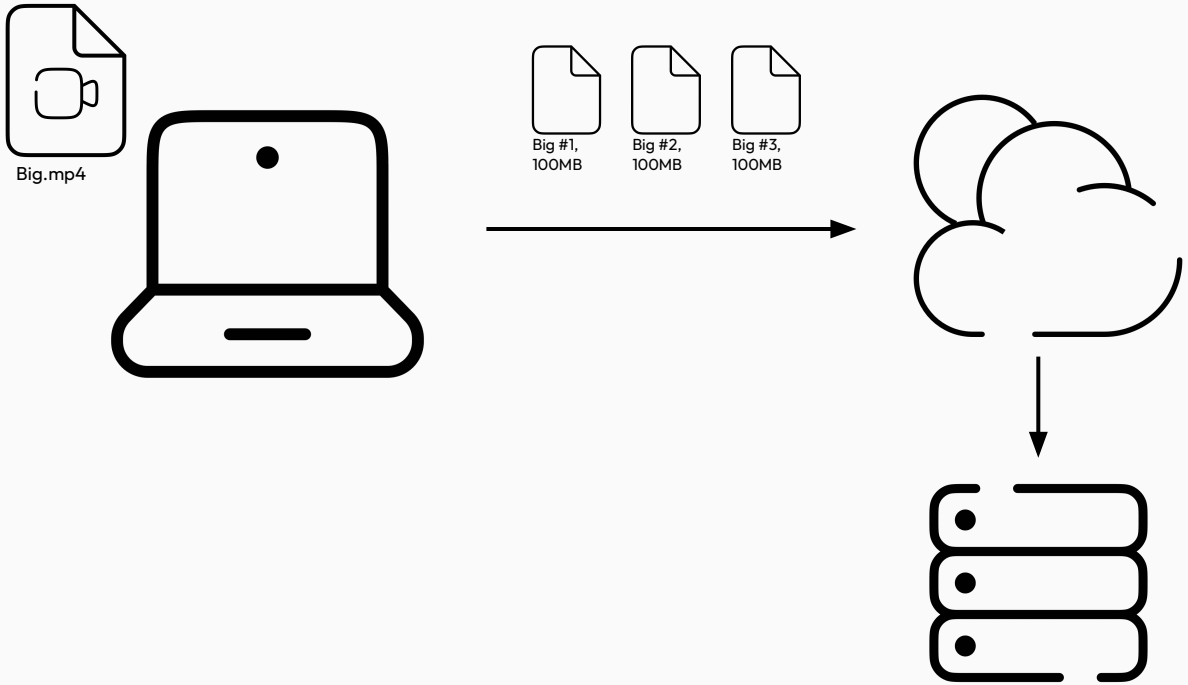
Chunking: Synchronization



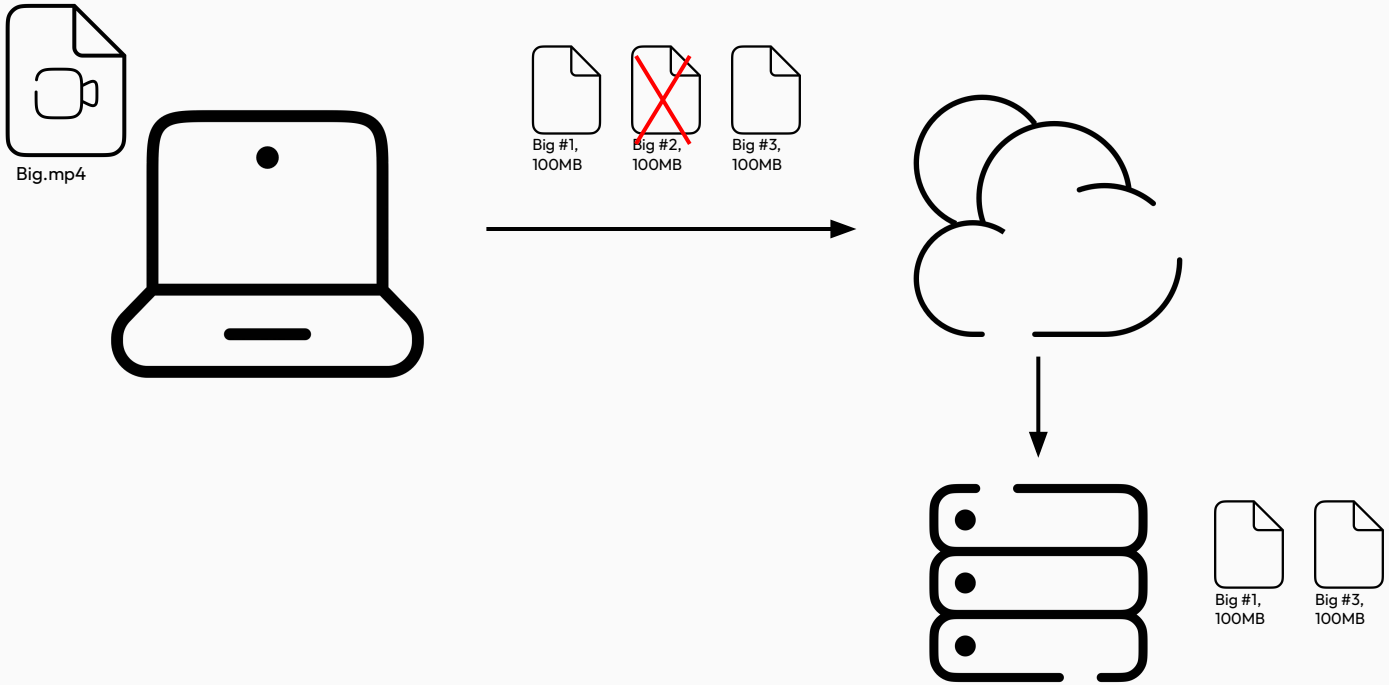
Chunking: Synchronization



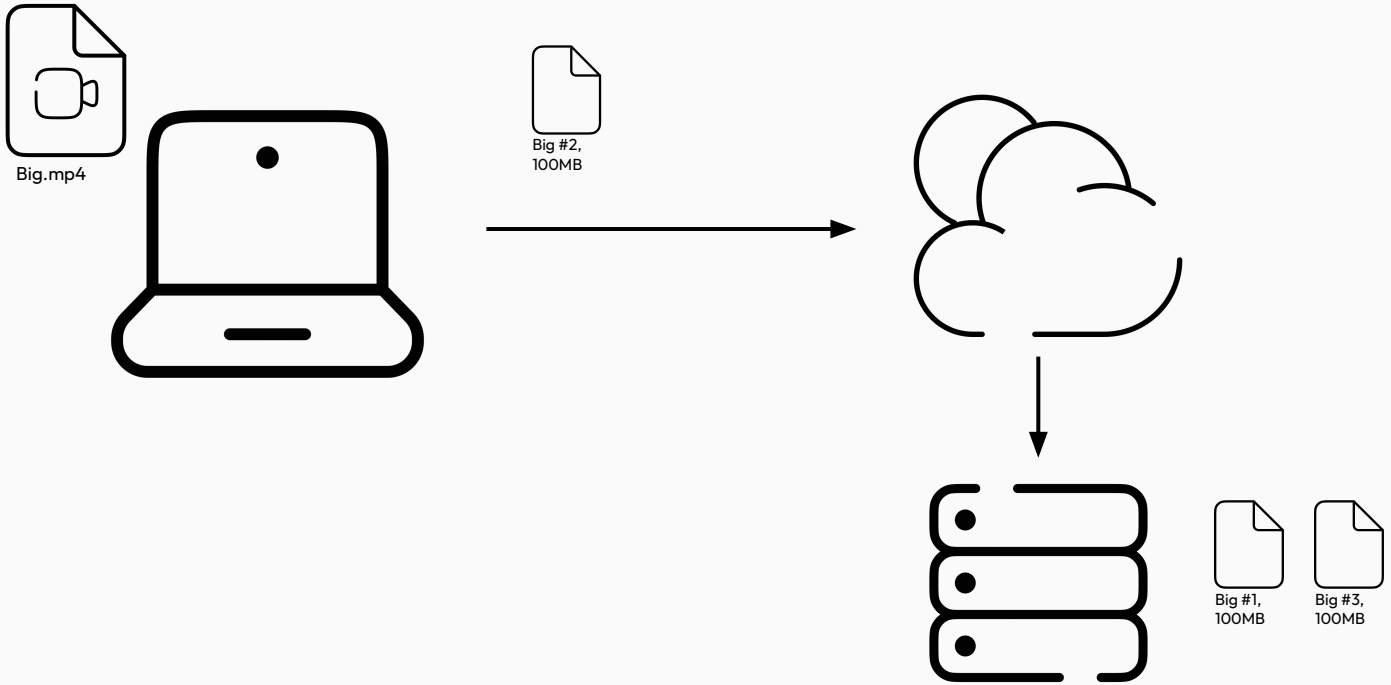
Chunking: Synchronization



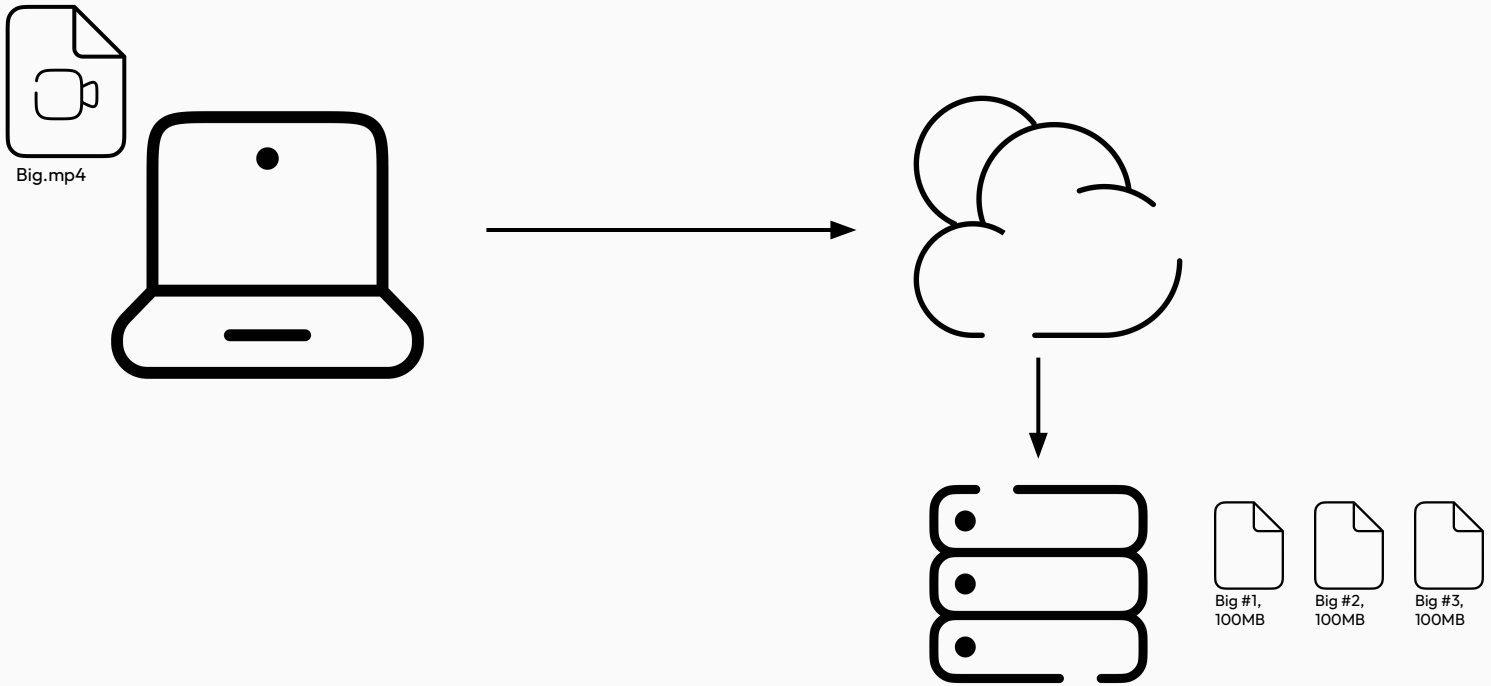
Chunking: Synchronization



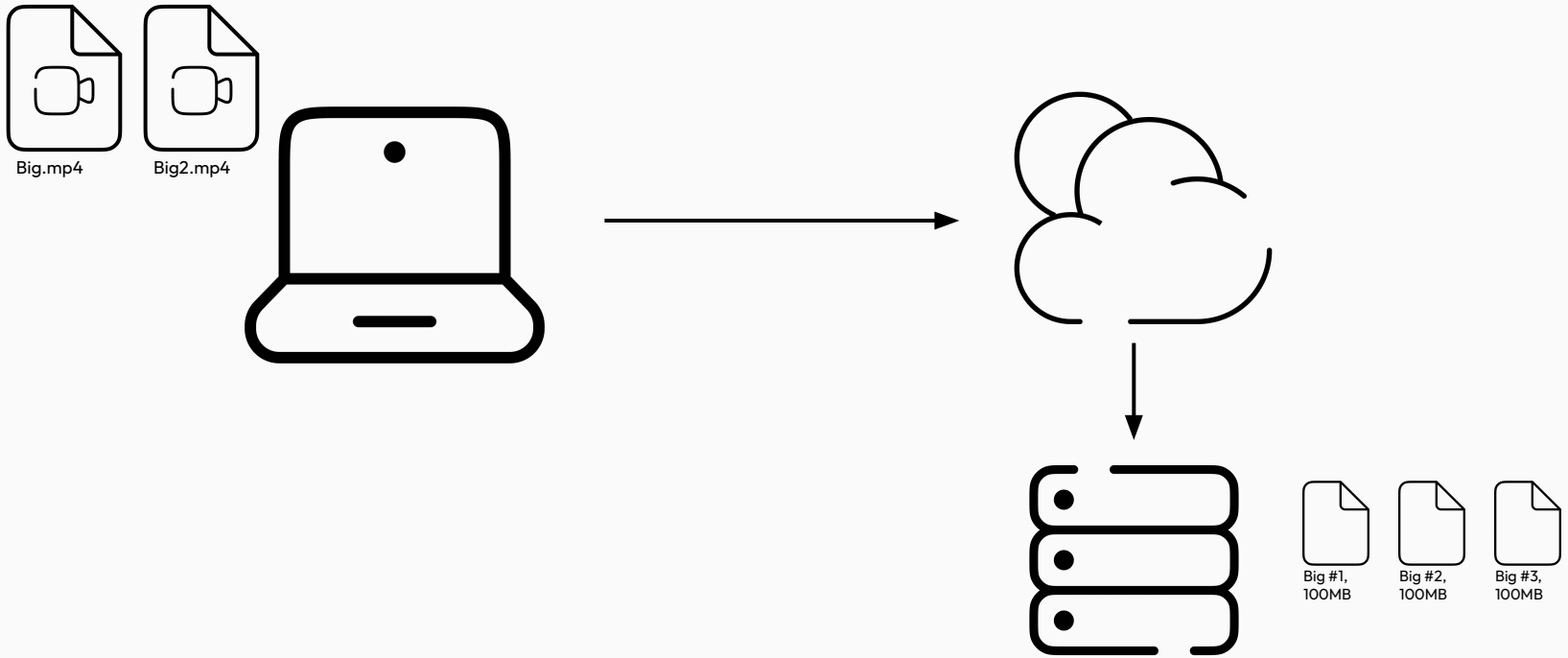
Chunking: Synchronization



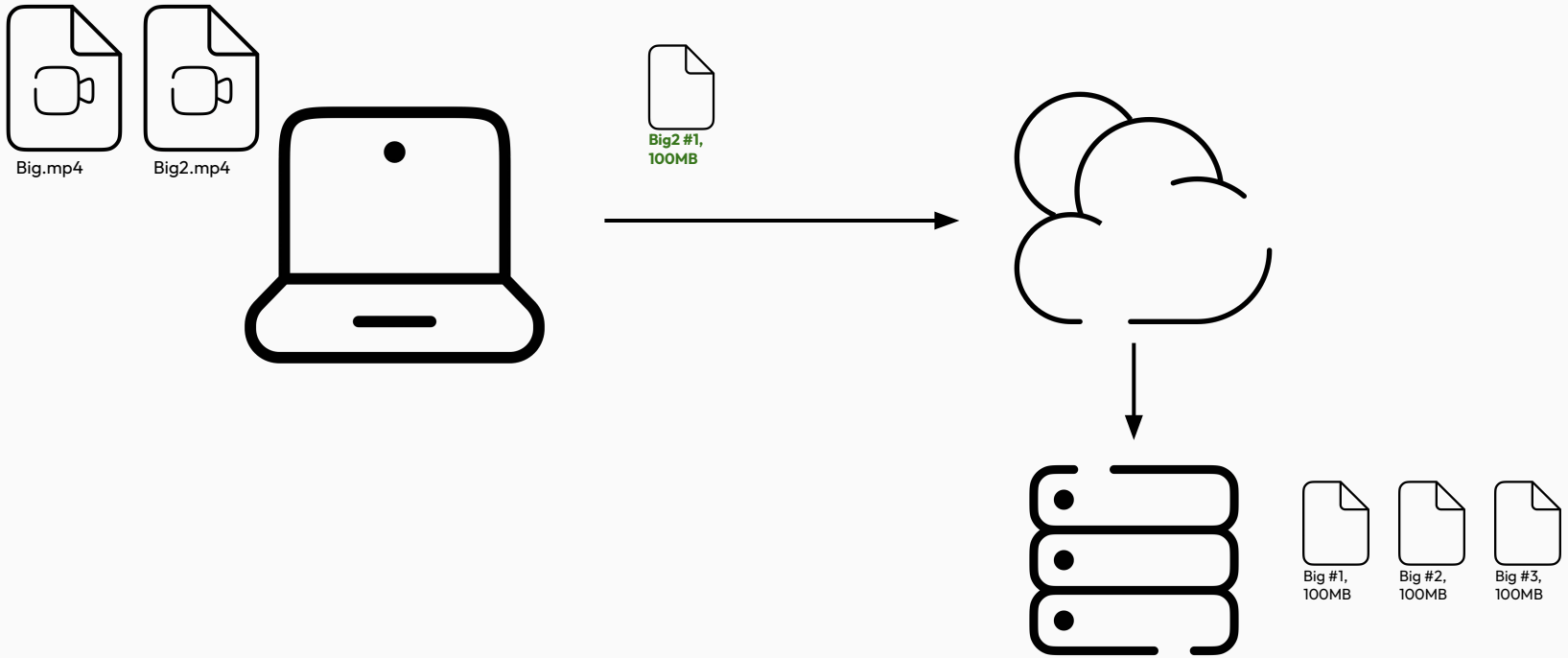
Chunking: Synchronization



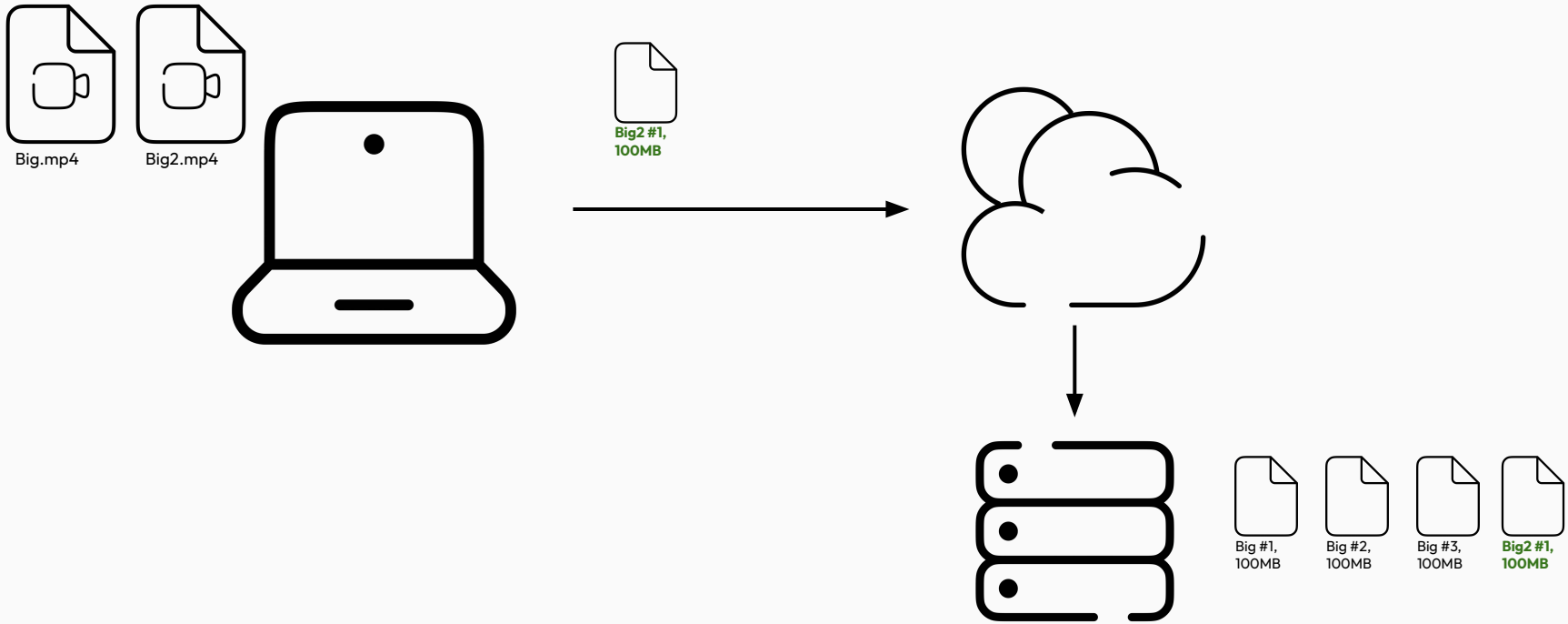
Chunking: Deduplication



Chunking: Deduplication



Chunking: Deduplication



Chunking is Everywhere

Chunking is Everywhere

Chunking is Everywhere

Chunking is Everywhere

Chunking is Everywhere

Chunking: How To?

veryverylongfilefil

ledwithrandomdatauy

lpfbrccfhuvfadhsrby

hcvgovhpkfshjupfjnz

slyprsjcsdrlimltvvh

whfddjmusnvvdjkz ...

?

Fixed--Size Chunking

veryverylongfilef**il**|
ledwithrandomdata**uy**|
lpfbrccfhuvfadhsr**by**|
hcvgovhpkfshjupfj**nz**|
slyprsjcsdrlimltv**vh**|
hwhfddjmusnvvdjkz ...

Fixed-Size Chunking

veryverylongfilef**il**|
ledwithrandomdata**uy**|
lpfbrccfhuvfadhsr**by**|
hcvgovhpkfshjupfj**nz**|
slyprsjcsdrlimltv**vh**|
hwhfddjmusnvvdjkz ...

veryverylongfilefil

ledwithrandomdatauy

lpfbrccfhuvfadhsrby

hcvgovhpkfshjupfjnz

...

Fixed-Size Chunking: Boundary Shift

averyverylongfilefi

lledwithrandomdatau

ylpfbrccfhuvfadhsrb

yhcvgovhpkfshjupfjn

zslyprsjcsdrlimltvv

hhwhfddjmusnvvdjk ...

Fixed-Size Chunking: Boundary Shift

averyverylongfilef**i**|

lledwithrandomdata**u**|

ylpfbrccfhuvfadhsr**b**|

yhcvgovhpkfshjupfj**n**|

zslyprsjcsdrlimltv**v**|

hhwhfddjmusnvvdjk ...

Fixed-Size Chunking: Boundary Shift

averyverylongfilefi|
lledwithrandomdatau|
ylpfbrccfhuvfadhsrb|
yhcvgovhpkfshjupfjn|
zslyprsjcsdrlimltvv|
hhwhfddjmusnvvdjk ...

Fixed-Size Chunking: Boundary Shift

averyverylongfilefi|
lledwithrandomdatau|
ylpfbrccfhuvfadhsrb|
yhcvgovhpkfshjupfjn|
zslyprsjcsdrlimltvv|
hhwhfddjmusnvvdjk ...

averyverylongfilefi

lledwithrandomdatau

ylpfbrccfhuvfadhsrb

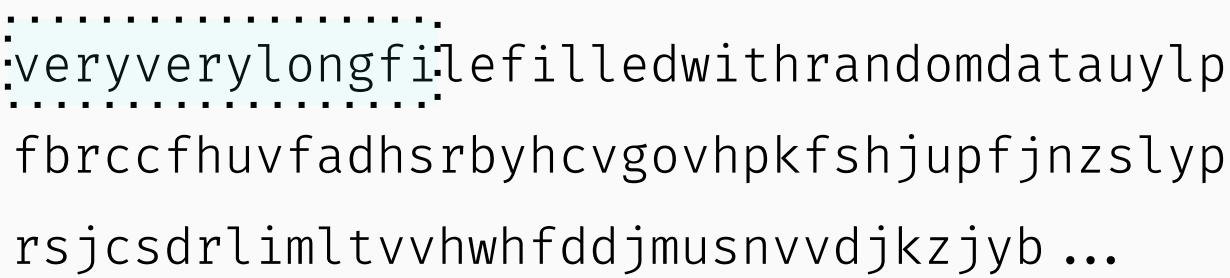
yhcvgovhpkfshjupfjn

...

Content-Defined?!?

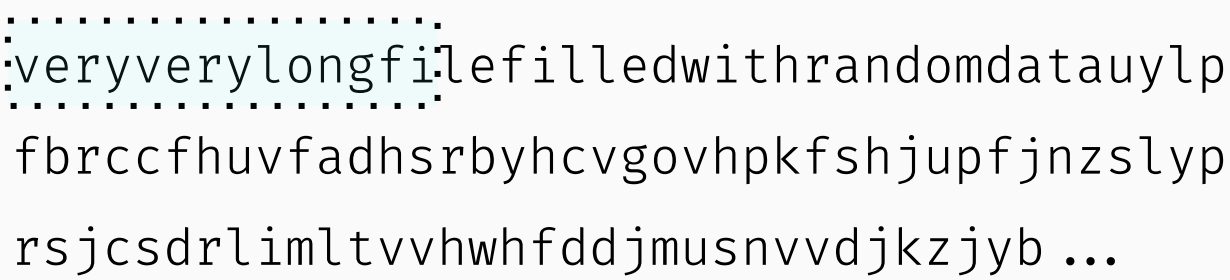
Content-Defined?!?

w = “very…”



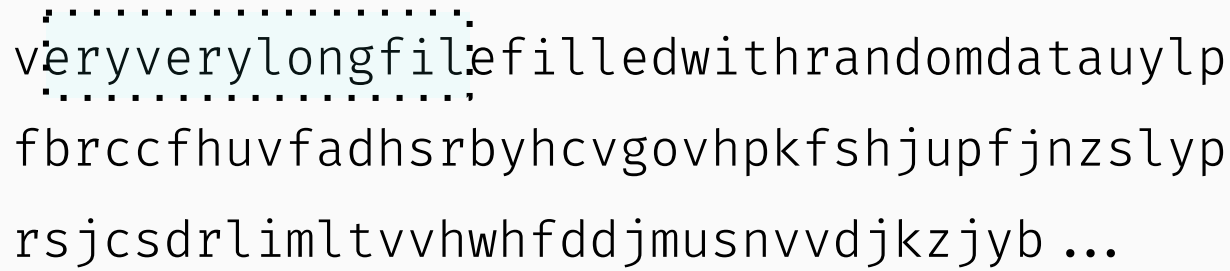
Content-Defined?!?

w = “very…” Π(w) == false



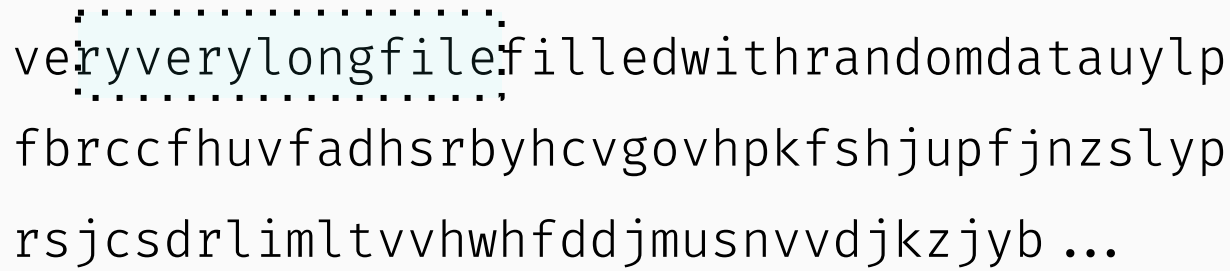
Content-Defined?!?

w = “ery…fil” Π(w) == false



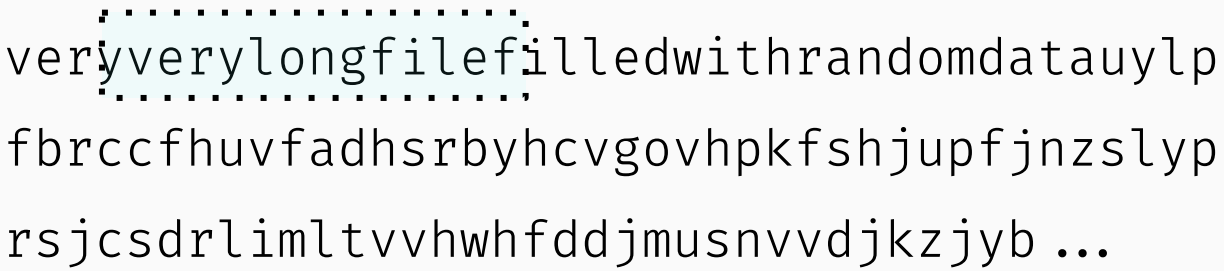
Content-Defined?!?

w = “ryv...ile” Π(w) == false



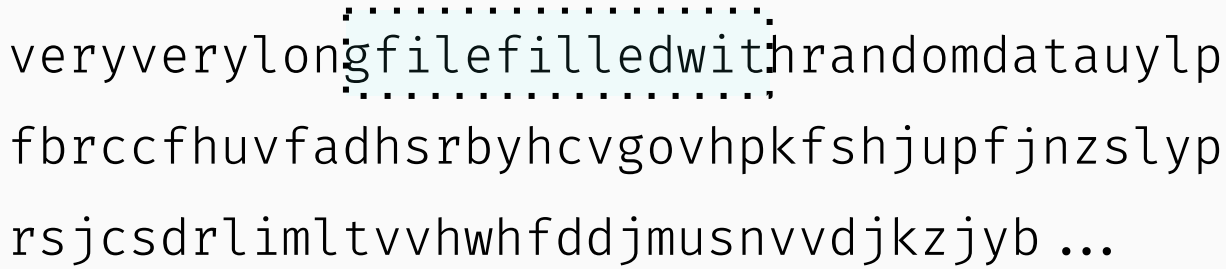
Content-Defined?!?

w = “yve…lef” Π(w) == false



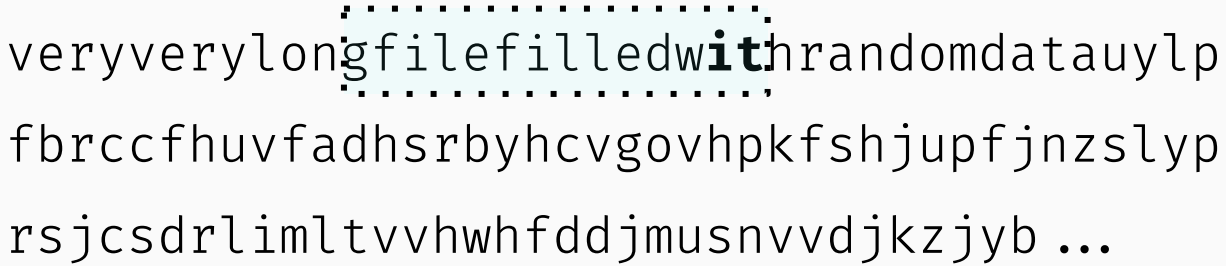
Content-Defined?!?

w = “gfi...wit”



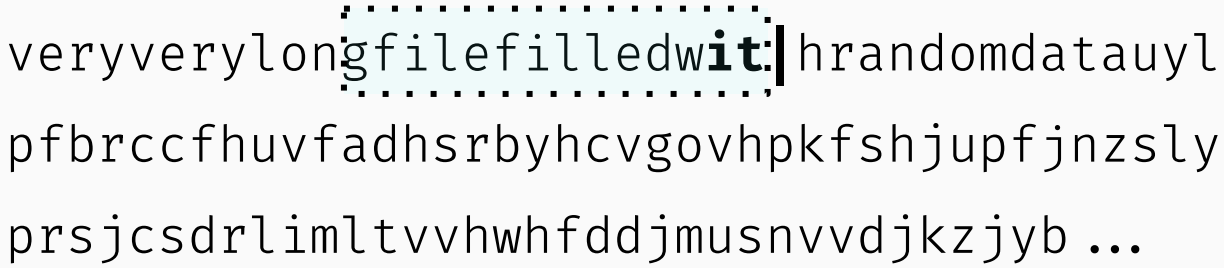
Content-Defined?!?

w = “gfi...wit” Π(w) == true



Content-Defined?!?

w = “gfi...wit” Π(w) == true



Content-Defined?!?

veryverylongfilefilledw**it**|hrandomdatauyl

p**fb**|rccfhuvfadhsrbyhcvgovhpkfshjupfjnzsl

yprsjcs**dr**|limltvvhwhfddjmusnvvdjkzj ...

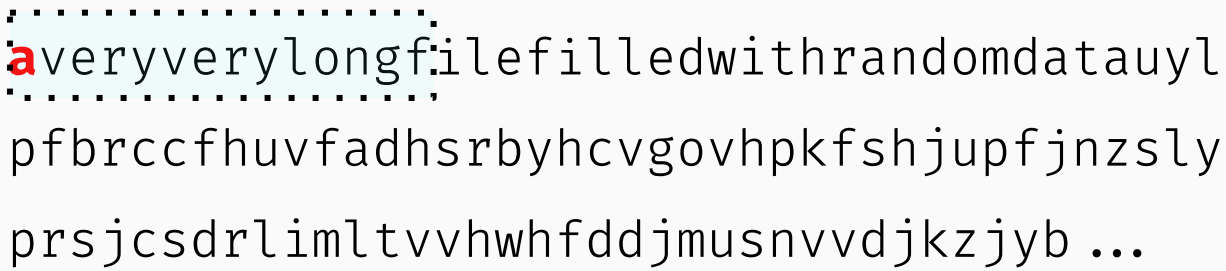
Content-Defined?!?

averyverylongfilefilledwithrandomdatauyl

pfbrccfhuvfadhsrbyhcvgovhpkfshjupfjnzsly

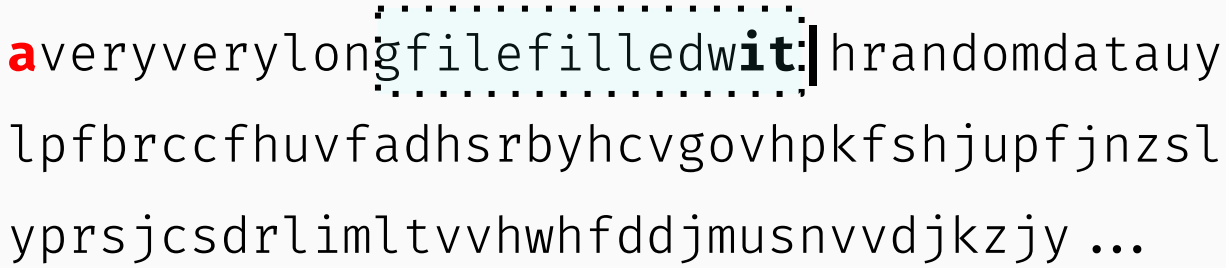
prsjcsdrlimltvvhwhfddjmusnvvdjkzjyb ...

Content-Defined?!?



Content-Defined?!?

Π(w) == true



Content-Defined Chunking

veryverylongfilefil

ledw**it**|hrandomdatau

ylp**fb**|rccfhuvfadhsr

byhcvgovhpkfshjupfj

nzslyprsjcs**dr**|limlt

vvhwhfddjmusnvvdjkz

...

veryverylongfilefilledw**it**

hrandomdatauylp**fb**

rccfhuvfadhsrbyhcvgovhpkfs

hjupfjnzslyprsjcs**dr**

...

Content-Defined Chunking

averyverylongfilefi
lledw**it**|hrandomdata
uylp**fb**|rccfhuvfadhs
rbyhcvgovhpkfshjupf
jnzslyprsjcs**dr**|liml
tvvhwhfddjmusnvvdjk
...

averyverylongfilefilledw**it**

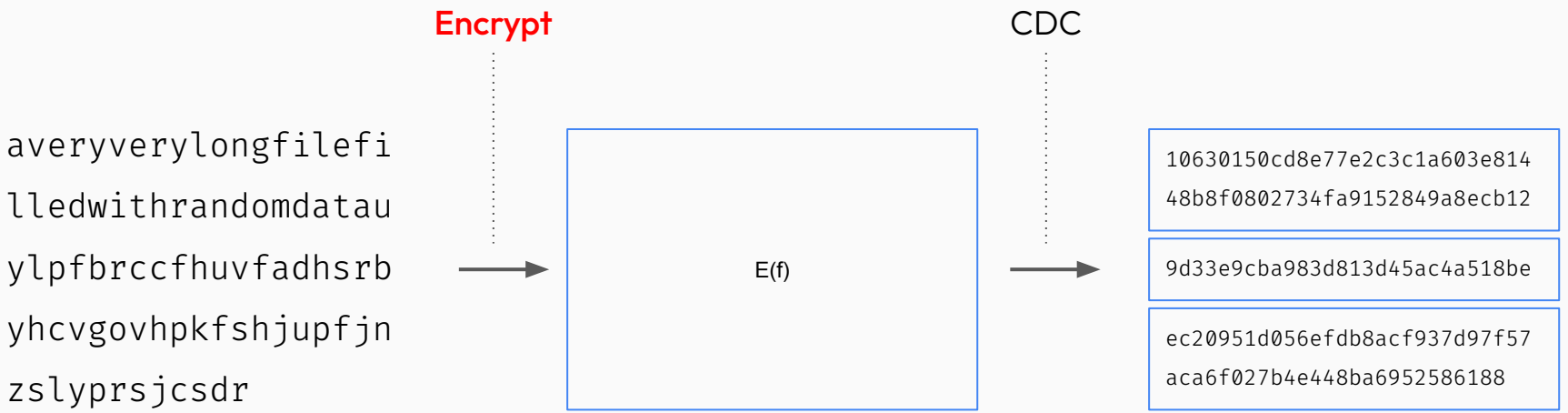
hrandomdatauylp**fb**

rccfhuvfadhsrbyhcvgovhpkfs
hjupfjnzslyprsjcs**dr**

...

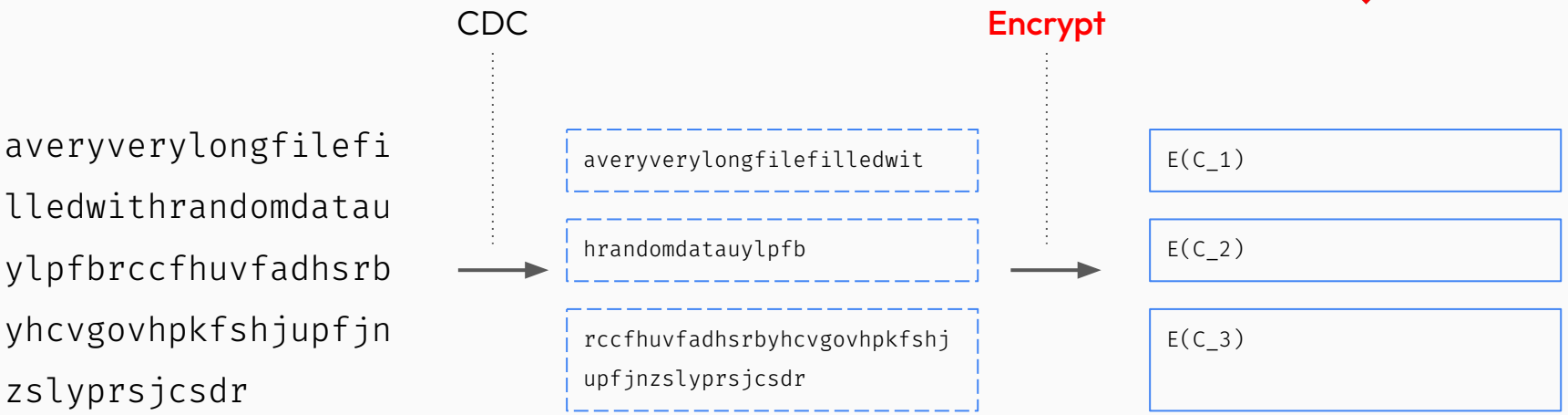
Photo by tommao wang on Unsplash

Content-Defined Chunking + Encryption

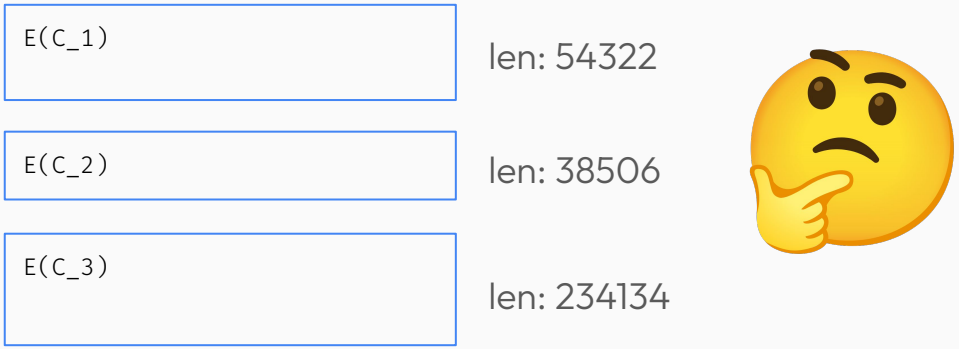


Content-Defined Chunking + Encryption

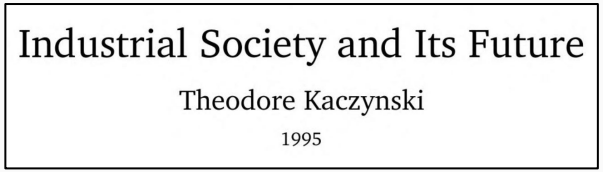
Content-Defined Chunking + Encryption



Fingerprinting Attack

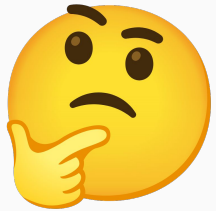


Fingerprinting Attack

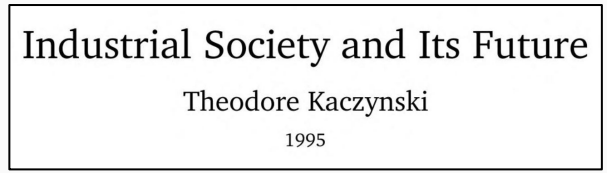


kaczynski2.pdf, 89.1 MB

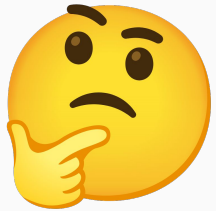
E(C_1)	len: 54322
E(C_2)	len: 38506
E(C_3)	len: 234134

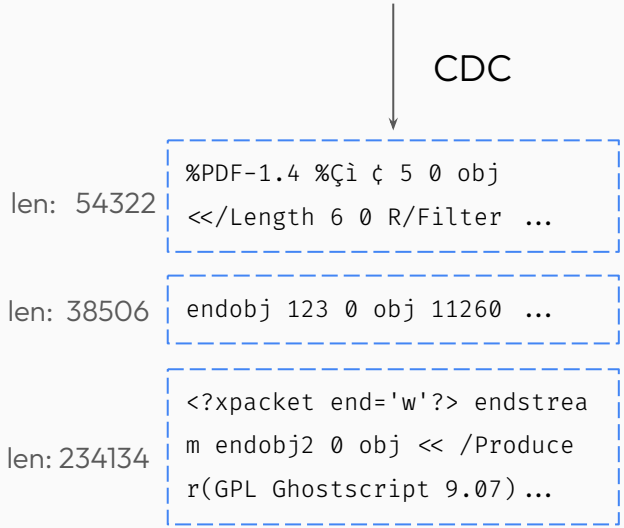


Fingerprinting Attack

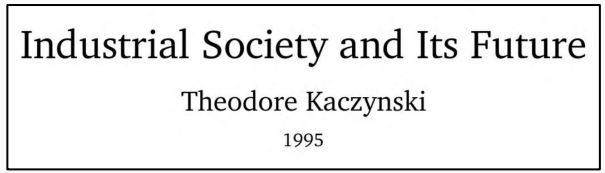


kaczynski2.pdf, 89.1 MB

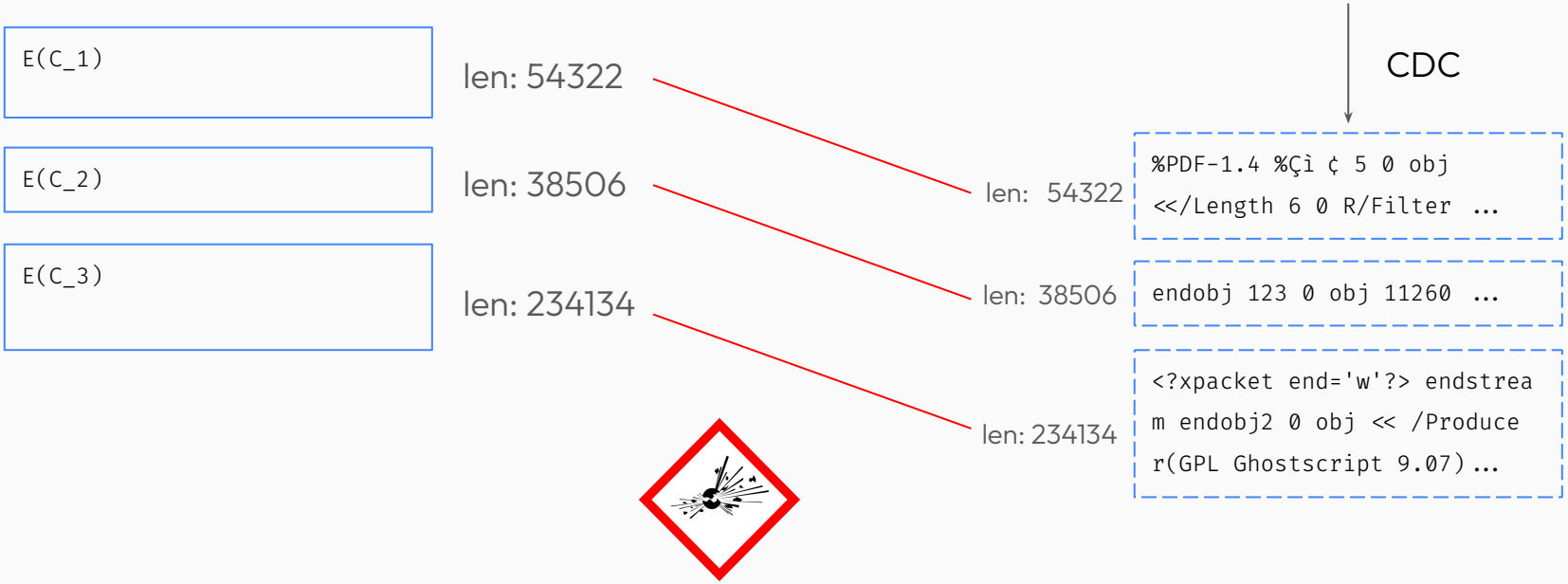




Fingerprinting Attack

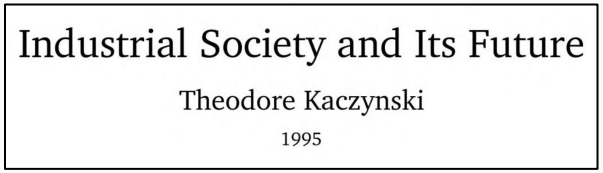


kaczynski2.pdf, 89.1 MB



Keyed Content-Defined Chunking

Keyed Content-Defined Chunking



kaczynski2.pdf, 89.1 MB

Chunking Key

K

KCDC

len: 54322

```
%PDF-1.4 %Çì ¢ 5 0 obj
<</Length 6 0 R/Filter ...
```

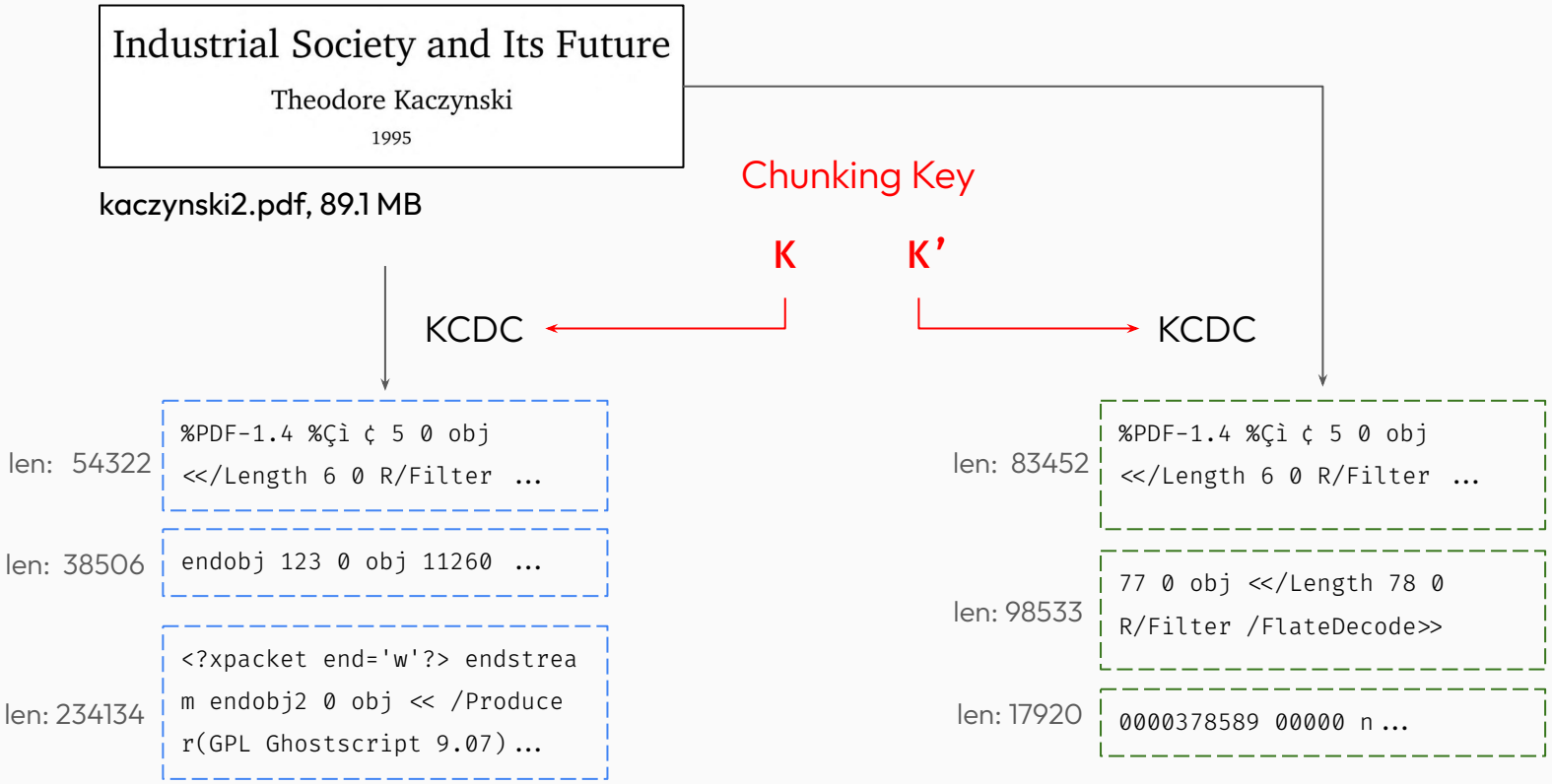
len: 38506

```
endobj 123 0 obj 11260 ...
```

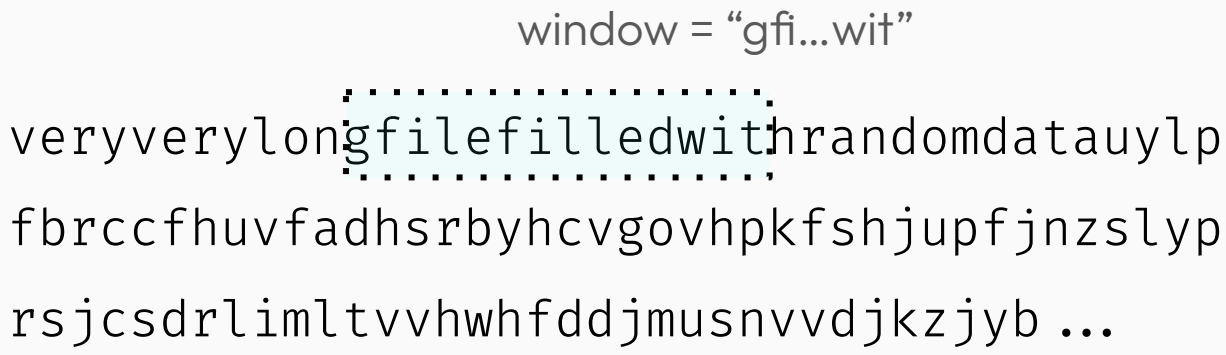
len: 234134

```
<?xpacket end='w'?> endstrea
m endobj2 0 obj << /Produce
r(GPL Ghostscript 9.07)...
```

Keyed Content-Defined Chunking

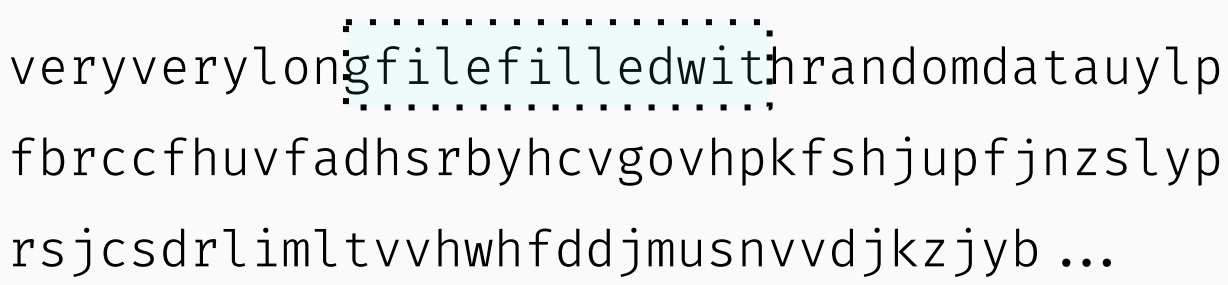


Keyed Content-Defined Chunking



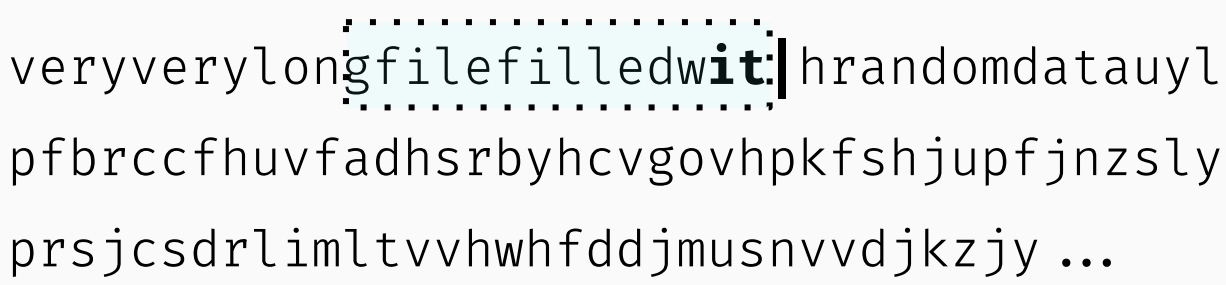
Keyed Content-Defined Chunking

KCDC.WindowEval(K, window) == true



Keyed Content-Defined Chunking

KCDC.WindowEval(K, window) == true



Keyed Content-Defined Chunking: Security

Keyed Content-Defined Chunking: Security

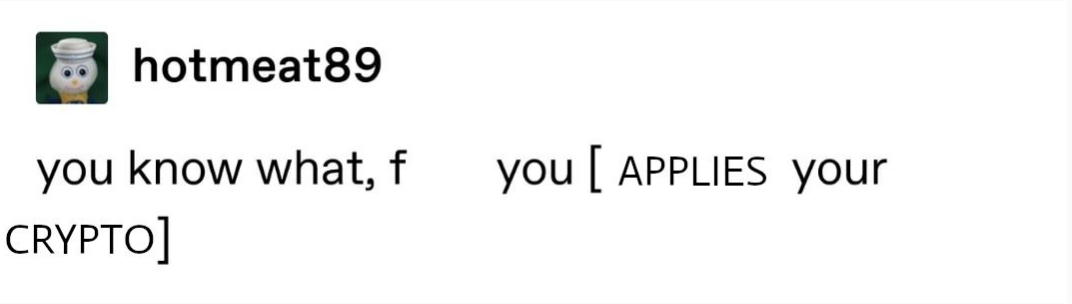
$$\text{KCDC.WindowEval}(K, \text{window}) == \text{true}$$

$$\approx$$

$$\text{Ber}(\lambda^{-1})$$

Average chunk size

TODO: slide for speaker transition



Desiderata

Applications require a KCDC that:

Desiderata

Applications require a KCDC that:

1. Preserves the “rolling” property

Desiderata

Applications require a KCDC that:

1. Preserves the “rolling” property
2. Is *very* cheap to evaluate

Desiderata

Applications require a KCDC that:

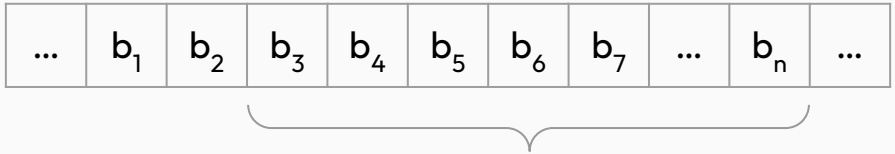
1. Preserves the “rolling” property
2. Is *very* cheap to evaluate

Eval has to be called for (almost) every byte of the file (~10^9 to ~10^{12} times)

KCDC, in the Wild

...	b_1	b_2	b_3	b_4	b_5	b_6	b_7	...	b_n	...

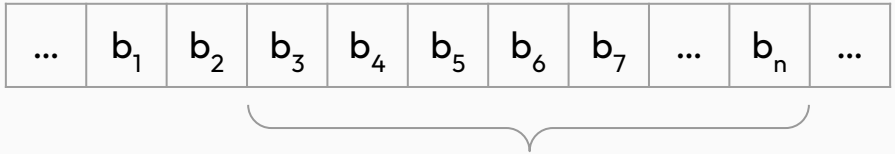
KCDC, in the Wild



Step 1

Compute a polynomial using “some window” of bytes

KCDC, in the Wild

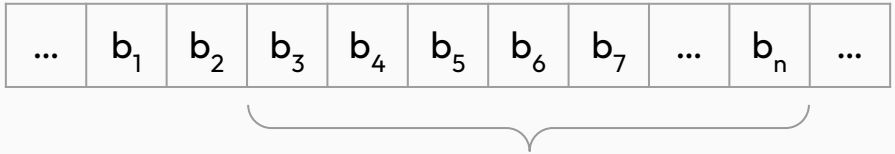


Step 1

Compute a polynomial using “some window” of bytes

This provides the rolling property

KCDC, in the Wild



Step 1

Compute a polynomial using “some window” of bytes

This provides the rolling property

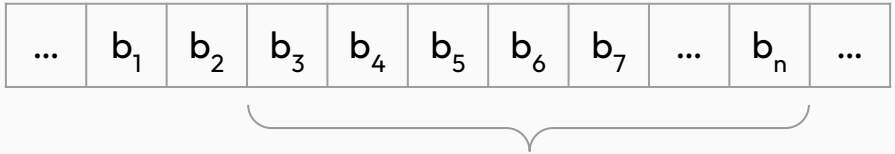
$Q(x)$

Step 2

Evaluate a predicate Π on the polynomial

$\Pi(Q(x)) = 1?$

KCDC, in the Wild



Step 1

Compute a polynomial using “some window” of bytes

This provides the rolling property

Q(x)

Step 2

Evaluate a predicate Π on the polynomial

Π(Q(x)) = 1?

Both steps may involve the chunking key

The Restic KCDC

The Restic KCDC

restic

Backups done right!

Let P be a *secret* irreducible polynomial over F_2

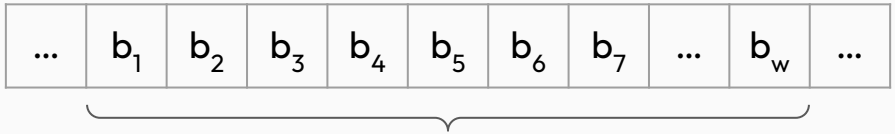
...	b_1	b_2	b_3	b_4	b_5	b_6	b_7	...	b_w	...

The Restic KCDC

restic

Backups done right!

Let P be a *secret* irreducible polynomial over F_2



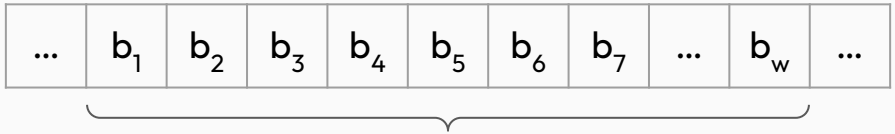
$Q(x) \in F_2[x]$, $\deg(Q) < 8w$

The Restic KCDC

restic

Backups done right!

Let P be a *secret* irreducible polynomial over F_2



$Q(x) \in F_2[x]$, $\deg(Q) < 8w$

If the last k coeffs. of Q(x) (mod P(x)) are 0, then chunk after b_w

$$Q(x) \pmod{P(x)} \pmod{x^k} = 0?$$

Attacking the Restic KCDC

$$Q(x) \pmod{P(x)} \pmod{x^k} = 0?$$

Attacking the Restic KCDC

$$Q(x) \pmod{P(x)} \pmod{x^k} = 0?$$

A polynomial Q(x) in a 8w-dimensional F_2-vector space...

Attacking the Restic KCDC

...after an (unknown) linear transformation L_p...

$$Q(x) \pmod{P(x)} \pmod{x^k} = 0?$$

A polynomial Q(x) in a 8w-dimensional F_2-vector space...

Attacking the Restic KCDC

…after an (unknown) linear transformation L_p…

$$Q(x) \pmod{P(x)} \pmod{x^k} = 0?$$

A polynomial Q(x) in a 8w-dimensional F_2-vector space…

…is mapped to 0

Attacking the Restic KCDC

…after an (unknown) linear transformation L_P…

$$Q(x) \pmod{P(x)} \pmod{x^k} = 0?$$

A polynomial Q(x) in a 8w-dimensional F_2-vector space…

…is mapped to 0

After 8w-k chunks, we obtain (whp.) a basis for $\mathrm{Ker}(L_P)$

Attacking the Restic KCDC

…after an (unknown) linear transformation L_P…

$$Q(x) \pmod{P(x)} \pmod{x^k} = 0?$$

A polynomial Q(x) in a 8w-dimensional F_2-vector space…

…is mapped to 0

After 8w-k chunks, we obtain (whp.) a basis for $\mathrm{Ker}(L_P)$

This is sufficient to efficiently recover P, allowing fingerprinting again

Attacking the Restic KCDC

…after an (unknown) linear transformation L_P…

$$Q(x) \pmod{P(x)} \pmod{x^k} = 0?$$

A polynomial Q(x) in a 8w-dimensional F_2-vector space…

…is mapped to 0

After 8w-k chunks, we obtain (whp.) a basis for $\mathrm{Ker}(L_P)$

This is sufficient to efficiently recover P, allowing fingerprinting again


bupstash.io
Tarsnap
Online backups for the truly paranoid
Duplicacy
restic
Backups done right!
Borg

The Chunkyard

Desiderata

Applications require a KCDC that:

1. Preserves the “rolling” property
2. Is *very* cheap to evaluate

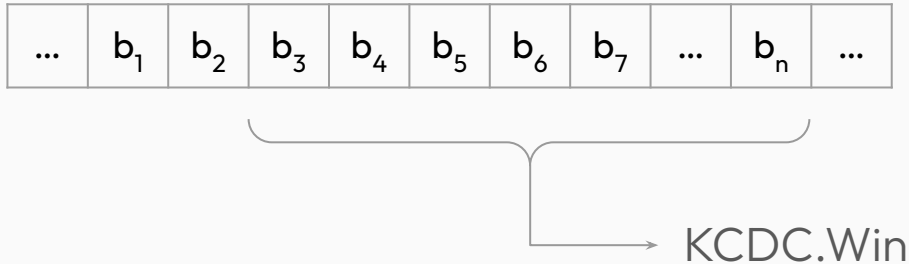
Desiderata

actually

Applications require a KCDC that:

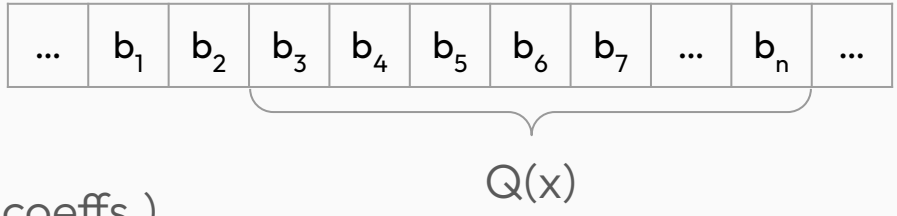
1. Preserves the “rolling” property
2. Is *very* cheap to evaluate
3. *Its chunking decisions are pseudorandom* (in the cryptographic sense)

The KCDC Security Notion



*Computationally, up to consistency of WindowEval

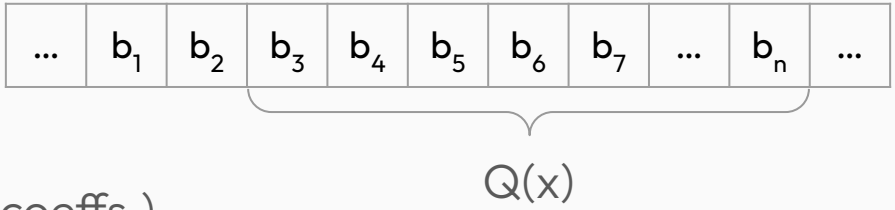
Polynomial+PRF Composition



Step 1

Compute a polynomial using w bytes (as coeffs.)

Polynomial+PRF Composition

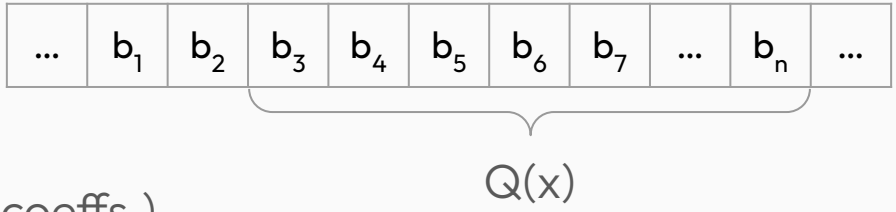


Step 1

Compute a polynomial using w bytes (as coeffs.)

$$Q(x)=b_3x^{w-1}+b_4x^{w-2}+\ldots+b_{n-1}x+b_n$$

Polynomial+PRF Composition

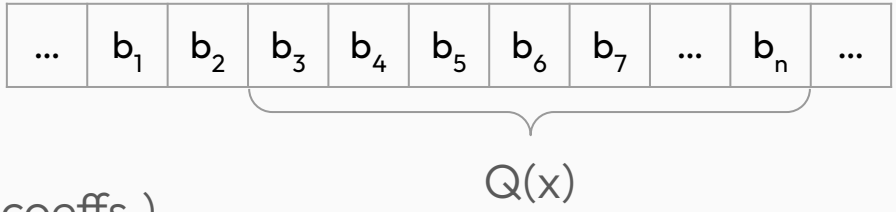


Step 1

Compute a polynomial using w bytes (as coeffs.)

$$Q'(x)=x(b_3x^{w-1}+b_4x^{w-2}+\ldots+b_{n-1}x+b_n)-b_3x^w+b_{n+1}$$

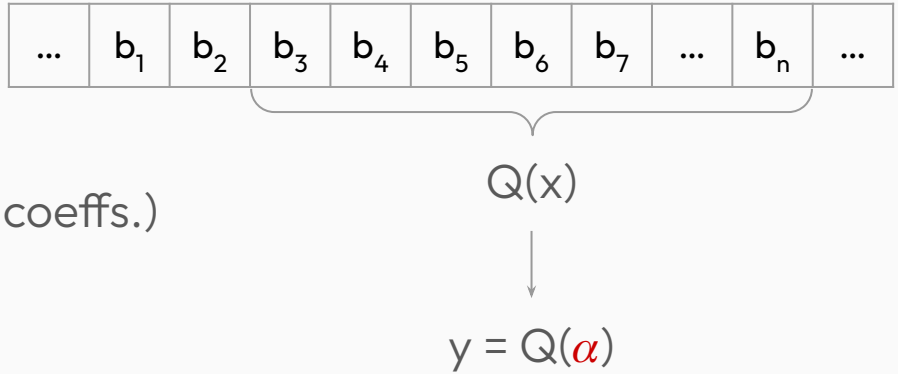
Polynomial+PRF Composition



Step 1
Compute a polynomial using w bytes (as coeffs.)

$$Q(x)=b_2x^{w-1}+b_3x^{w-2}+\ldots+b_{n-1}x+b_n$$

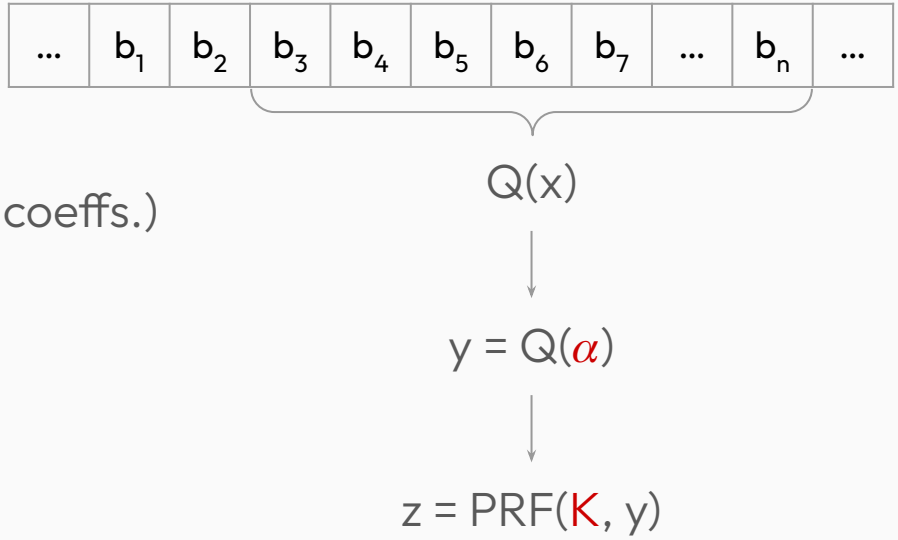
Polynomial+PRF Composition



Step 1
Compute a polynomial using w bytes (as coeffs.)

Step 2
Evaluate Q(x) at a secret point α

Polynomial+PRF Composition



Step 1
Compute a polynomial using w bytes (as coeffs.)

Step 2
Evaluate Q(x) at a secret point α

Step 3
Evaluate PRF(K, y)

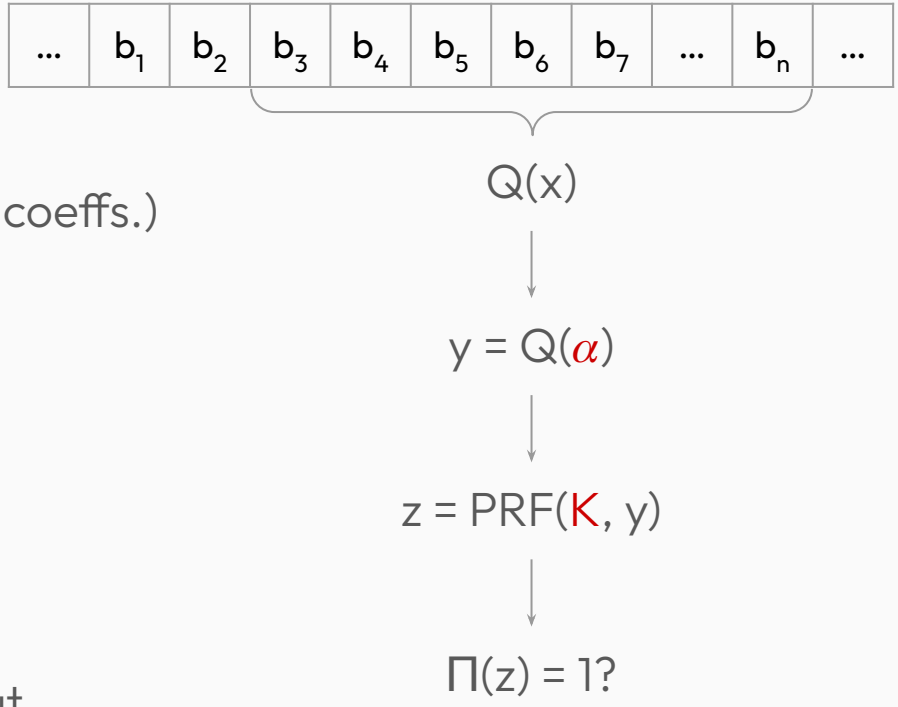
Polynomial+PRF Composition

Step 1
Compute a polynomial using w bytes (as coeffs.)

Step 2
Evaluate Q(x) at a secret point α

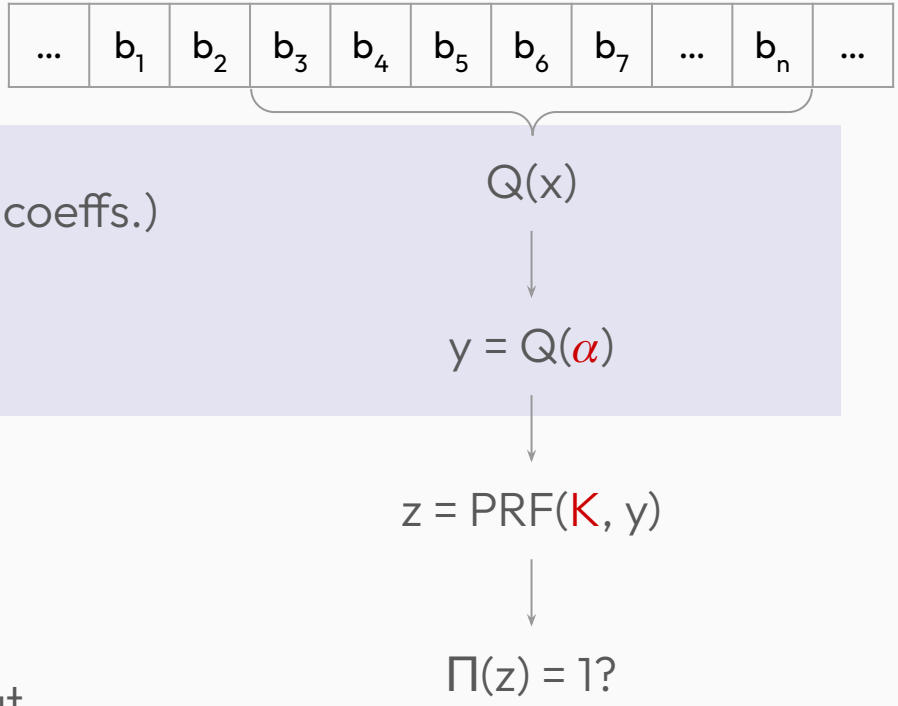
Step 3
Evaluate PRF(K, y)

Step 4
Evaluate a predicate Π on the PRF output



Polynomial+PRF Composition

UHF Evaluation



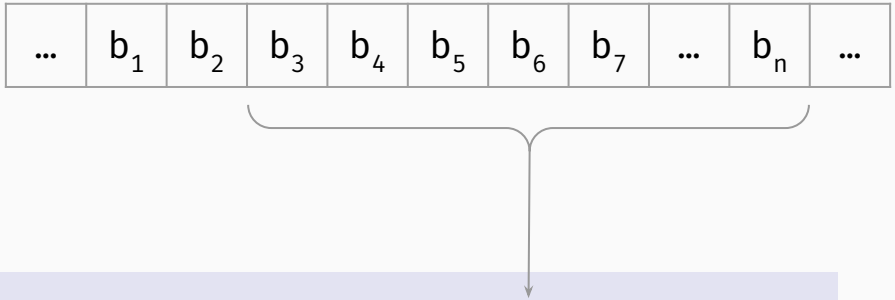
Step 1
Compute a polynomial using w bytes (as coeffs.)

Step 2
Evaluate Q(x) at a secret point α

Step 3
Evaluate PRF(K, y)

Step 4
Evaluate a predicate Π on the PRF output

(Rolling) UHF+PRF Composition



Step 1
Compute a (rolling) UHF over the window

$y = UHF(K_{UHF},(b_3,\ldots,b_n))$

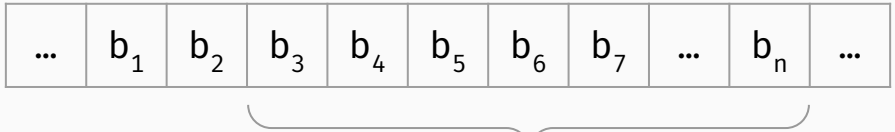
Step 2
Evaluate $PRF(K_{PRF}, y)$

$z = PRF(K_{PRF}, y)$

Step 3
Evaluate a predicate Π on the PRF output

$\Pi(z) = 1?$

(Rolling) UHF+PRF Composition



We prove that Restic's KCDC is a UHF!

Step 1
Compute a (rolling) UHF over the window

$y = UHF(K_{UHF},(b_3,\ldots,b_n))$

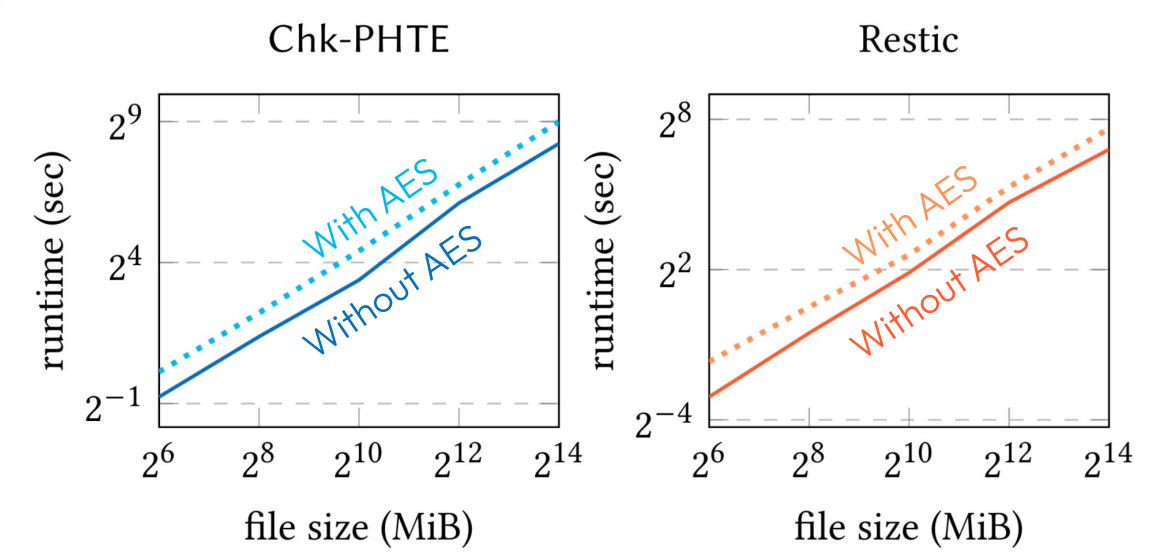
Step 2
Evaluate $PRF(K_{PRF}, y)$

$z = PRF(K_{PRF}, y)$

Step 3
Evaluate a predicate Π on the PRF output

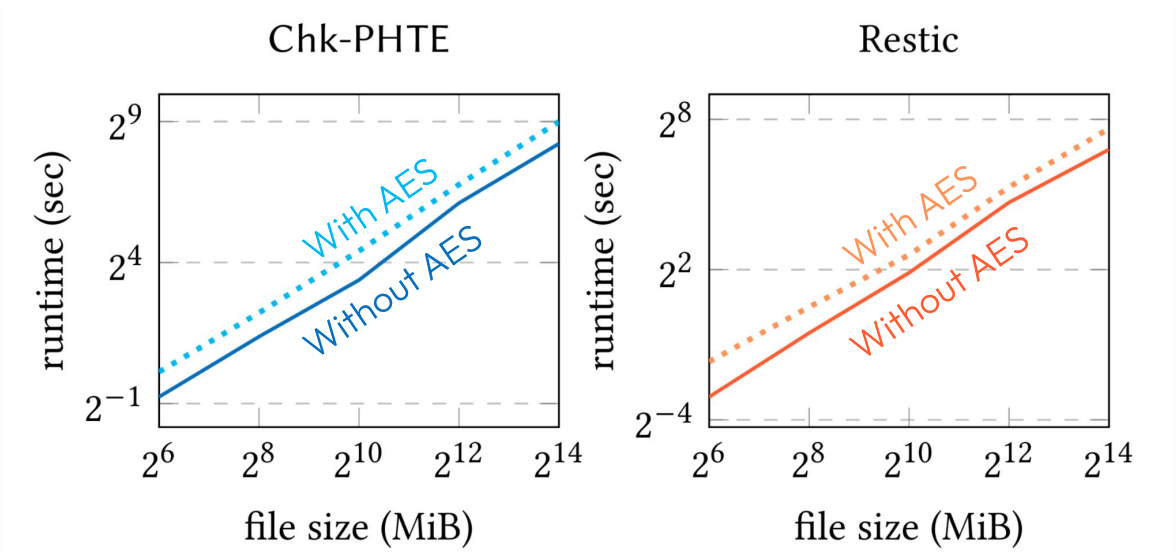
$\Pi(z) = 1?$

Performance



1.5x to 2x slowdown!

Performance



1.5x to 2x slowdown!

But is this even necessary to prevent fingerprinting?

Attacking the Restic KCDC

…after an (unknown) linear transformation L_P…

$$Q(x) \pmod{P(x)} \pmod{x^k} = 0?$$

A polynomial Q(x) in a 8w-dimensional F_2-vector space…

…is mapped to 0

After 8w-k chunks, we obtain (whp.) a basis for $\mathrm{Ker}(L_P)$

This is sufficient to efficiently recover P, allowing fingerprinting again

Attacking the Restic KCDC

…after an (unknown) linear transformation L_P…

$$Q(x) \pmod{P(x)} \pmod{x^k} = 0?$$

A polynomial Q(x) in a 8w-dimensional F_2-vector space…

…is mapped to 0

After 8w-k chunks, we obtain (whp.) a basis for $\mathrm{Ker}(L_P)$

This is sufficient to efficiently recover P, allowing fingerprinting again

This only works if we have *certain* knowledge of the chunk boundaries

The Chunkyard

Did we break *the KCDC* or *the backup system?*

The Chunkyard

Did we break the KCDC or *the backup system?*

The Chunkyard

Alternative Mitigations?

- Padding twarths our attacks in practice, so it *may* be a sufficient mitigation

Alternative Mitigations?

- Padding twarths our attacks in practice, so it *may* be a sufficient mitigation
- However, we cannot capture this in our security model

Alternative Mitigations?

- Padding twarths our attacks in practice, so it *may* be a sufficient mitigation
- However, we cannot capture this in our security model
- An open question for the community:

Alternative Mitigations?

- Padding twarths our attacks in practice, so it *may* be a sufficient mitigation
- However, we cannot capture this in our security model
- An open question for the community:

> *"Can we come up with a security model for efficient (!) chunk-based deduplication that allows efficient constructions, and captures processes like compression, padding, packing of chunks?"*

Alternative Mitigations?

- Padding twarths our attacks in practice, so it *may* be a sufficient mitigation
- However, we cannot capture this in our security model
- An open question for the community:

> *"Can we come up with a security model for efficient (!) chunk-based deduplication that allows efficient constructions, and captures processes like compression, padding, packing of chunks?"*

- Another (open) question for the community:

Alternative Mitigations?

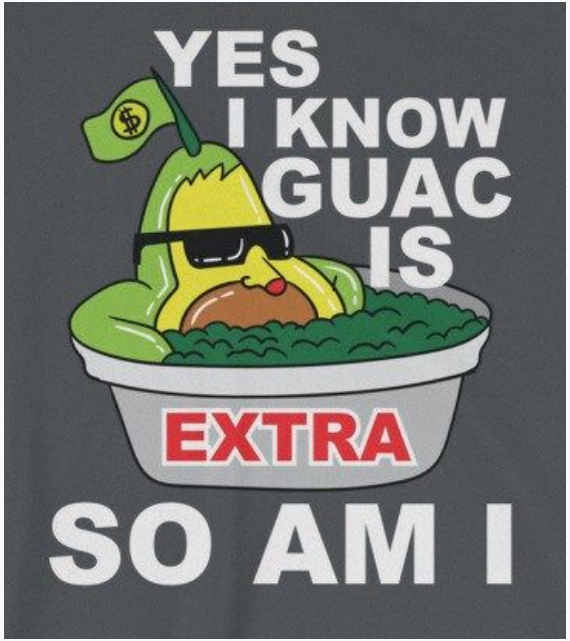
- Padding twarths our attacks in practice, so it *may* be a sufficient mitigation
- However, we cannot capture this in our security model
- An open question for the community:

> *"Can we come up with a security model for efficient (!) chunk-based deduplication that allows efficient constructions, and captures processes like compression, padding, packing of chunks?"*

- Another (open) question for the community:

> *"Does anyone want some empty cardboard boxes for free?"*

EXTRA SLIDES

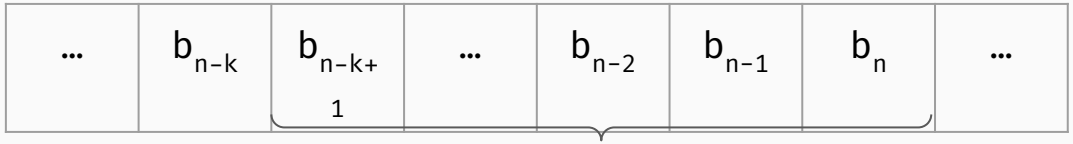


TODO: remove bad memes

The Tarsnap KCDC

Secret chunking key:

- p, a prime
- α, an element of "large" order in $\mathbb{Z}/p\mathbb{Z}$
- ϕ, a mapping $\{0,1\}^8 \to \mathbb{Z}/p\mathbb{Z}$



$$\text{Let } Q_{k,n}(x) = \phi(b_{n-k+1})x^k + \ldots + \phi(b_{n-1})x + \phi(b_n) \in (\mathbb{Z}/p\mathbb{Z})[x]$$

k: *relation length*

If there **exists** a k such that $Q_{k,n}(\alpha) = 0$, chunk after b_n

The Tarsnap KCDC: Degrees of Uncertainty

Because k is unknown, each chunk provides *uncertain* information

$$\phi(b_{n-31})\alpha^{32} + \ldots + \phi(b_{n-1})\alpha + \phi(b_n) = 0?$$

or

$$\phi(b_{n-32})\alpha^{33} + \phi(b_{n-31})\alpha^{32} + \ldots + \phi(b_{n-1})\alpha + \phi(b_n) = 0?$$

or

$$\phi(b_{n-33})\alpha^{34} + \phi(b_{n-32})\alpha^{33} + \phi(b_{n-31})\alpha^{32} + \ldots + \phi(b_{n-1})\alpha + \phi(b_n) = 0?$$

...

The Tarsnap KCDC: Recovering p, α, ϕ

Given many polynomials in $\mathbb{Z}/p\mathbb{Z}$ of *unknown* degrees $(k_1, \ldots, k_m)$, that evaluate to 0 at an *unknown* point α, with *unknown* coefficients determined by ϕ, find (p, α, ϕ)

- What helps us:
  - p has only 17 possible values
  - $Q_{k,n}(\alpha) = 0 \Leftrightarrow cQ_{k,n}(\alpha) = 0$, so ϕ is defined up to a scalar c
  - The relation lengths are upper bounded by ~1000
- Attack idea: set $\phi(0) = 1$, and recover $\phi(1)$ using a (chosen) plaintext composed only of 0 and 1 bytes.

The Tarsnap KCDC: Attack 1

Consider two polynomials R(x), S(x), known up to their degrees k_1, k_2 involving only bytes with values 0 or 1

$$R(x) = \phi(0) \cdot R_0(x) + \phi(1) \cdot R_1(x)$$

$$S(x) = \phi(0) \cdot S_0(x) + \phi(1) \cdot S_1(x)$$

Bruteforce the values of p, k_1, k_2 determining $R_0(x)$, $R_1(x)$, $S_0(x)$, $S_1(x)$.

Then, α must be a root of $R_0(x)S_1(x)-S_0(x)R_1(x)$.

Finally, use linear algebra to retrieve $\phi(1)$.

Takeaways

Current KCDC constructions leak information about their keys *because they have too much algebraic structure*.

Takeaways

Current KCDC constructions leak information about their keys *because they have too much algebraic structure*.

However, *algebraic structure is needed* to efficiently “roll” the hash one byte forward.

Takeaways

Current KCDC constructions leak information about their keys *because they have too much algebraic structure*.

However, *algebraic structure is needed* to efficiently “roll” the hash one byte forward.

Can we do better?

A diagonalized basis of $\mathrm{Ker}(L_P)$ has polynomials of all degrees between 8w-1 to k+1.

Consider a basis polynomial of degree d, B(x).

Then, $B(x) + P(x) = 0 \pmod{x^{20}} \Rightarrow B(x) \pmod{x^{20}} = P(x) \pmod{x^{20}}$

Hence B(x) leaks the 20 least-significant coefficients of P(x).