Randomized Algorithms

Universal hashing: what randomness does lookup actually need?

Alex Gittens
Department of Computer Science, Rensselaer Polytechnic Institute

Fall 2026

Learning Objectives

By the end of this lecture, you should be able to:

1. Identify which properties of the randomness are actually used in a randomized analysis.
2. Construct universal hash families with short random seeds, including a Toeplitz construction that reuses random bits.
3. Distinguish guarantees for fixed inputs from guarantees that remain valid after conditioning on previous random choices.

How much randomness does hashing need?

We store a fixed set

$$S = \{y_1, \ldots, y_n\} \subseteq U$$

in a hash table with m buckets.

A convenient idealization is to choose a fully random function

$$h : U \to [m].$$

How much randomness does hashing need?

We store a fixed set

$$S = \{y_1, \ldots, y_n\} \subseteq U$$

in a hash table with m buckets.

A convenient idealization is to choose a fully random function

$$h : U \to [m].$$

If $|U| = N$, how many random bits specify such a function?

How much randomness does hashing need?

We store a fixed set

$$S = \{y_1, \ldots, y_n\} \subseteq U$$

in a hash table with m buckets.

A convenient idealization is to choose a fully random function

$$h : U \to [m].$$

If $|U| = N$, how many random bits specify such a function?
Each key needs one bucket label, so the answer is

$$N \log_2 m$$

random bits.

How much randomness does hashing need?

We store a fixed set

$$S = \{y_1, \ldots, y_n\} \subseteq U$$

in a hash table with m buckets.

A convenient idealization is to choose a fully random function

$$h : U \to [m].$$

If $|U| = N$, how many random bits specify such a function?
Each key needs one bucket label, so the answer is

$$N \log_2 m$$

random bits.

Cost of a fully random function

For 64-bit keys and about 2^{20} buckets, a literal random-function table needs

$$2^{64} \cdot 20 \text{ bits} \approx 4.6 \times 10^{19} \text{ bytes.}$$

That is tens of exabytes just to describe the hash function.

How much randomness does hashing need?

We store a fixed set

$$S = \{y_1, \ldots, y_n\} \subseteq U$$

in a hash table with m buckets.

A convenient idealization is to choose a fully random function

$$h : U \to [m].$$

If $|U| = N$, how many random bits specify such a function?
Each key needs one bucket label, so the answer is

$$N \log_2 m$$

random bits.

Cost of a fully random function

For 64-bit keys and about 2^{20} buckets, a literal random-function table needs

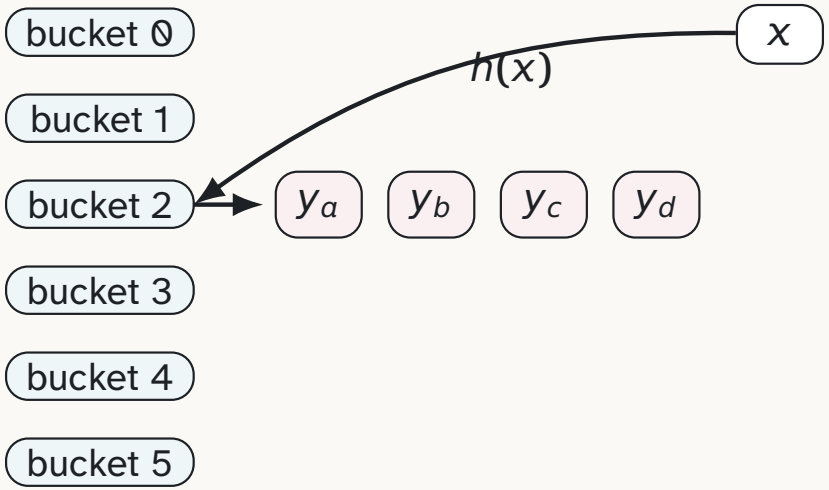
$$2^{64} \cdot 20 \text{ bits} \approx 4.6 \times 10^{19} \text{ bytes.}$$

That is tens of exabytes just to describe the hash function.

So what part of full randomness does the lookup analysis actually need?

Lookup cost with chaining

A chained hash table stores all keys with the same hash value in the same bucket.



For an unsuccessful query $x \notin S$, the lookup scans exactly the stored keys in bucket $h(x)$.

False matches

For a fixed unsuccessful query $x \notin S$, define

$$I_y = \mathbf{1}_{\{h(y)=h(x)\}}, \qquad C_x = \sum_{y \in S} I_y.$$

So C_x is the number of false matches scanned during the lookup.

False matches

For a fixed unsuccessful query $x \notin S$, define

$$I_y = \mathbf{1}_{\{h(y)=h(x)\}}, \qquad C_x = \sum_{y \in S} I_y.$$

So C_x is the number of false matches scanned during the lookup.

Thus the lookup cost is the number of collisions with x:

$$\mathbb{E} C_x = \mathbb{E} \sum_{y \in S} I_y$$

False matches

For a fixed unsuccessful query $x \notin S$, define

$$I_y = \mathbf{1}_{\{h(y)=h(x)\}}, \qquad C_x = \sum_{y \in S} I_y.$$

So C_x is the number of false matches scanned during the lookup.

Thus the lookup cost is the number of collisions with x:

$$\mathbb{E}C_x = \mathbb{E} \sum_{y \in S} I_y$$

$$= \sum_{y \in S} \mathbb{E} I_y = \sum_{y \in S} \mathbb{P}(h(y) = h(x)).$$

The fully random proof

If $h : U \to [m]$ is a fully random function, then every key receives an independent uniform bucket.

The fully random proof

If $h : U \to [m]$ is a fully random function, then every key receives an independent uniform bucket.
For every fixed stored key $y \neq x$,

$$\mathbb{P}(h(y) = h(x)) = \frac{1}{m}.$$

The fully random proof

If $h : U \to [m]$ is a fully random function, then every key receives an independent uniform bucket.
For every fixed stored key $y \neq x$,

$$\mathbb{P}(h(y) = h(x)) = \frac{1}{m}.$$

Therefore

$$\mathbb{E}C_x = \sum_{y \in S} \mathbb{P}(h(y) = h(x)) = \frac{n}{m}.$$

The fully random proof

If $h : U \to [m]$ is a fully random function, then every key receives an independent uniform bucket.
For every fixed stored key $y \neq x$,

$$\mathbb{P}(h(y) = h(x)) = \frac{1}{m}.$$

Therefore

$$\mathbb{E}C_x = \sum_{y \in S} \mathbb{P}(h(y) = h(x)) = \frac{n}{m}.$$

For $m \geq n$, an unsuccessful lookup scans $O(1)$ stored keys in expectation.

We now ask whether the same bound holds using a much shorter random seed.

Where did independence enter?

We wrote

$$\mathbb{E}C_x = \mathbb{E}\sum_{y\in S} I_y = \sum_{y\in S}\mathbb{E}I_y.$$

Question: where did we use independence?

Where did independence enter?

We wrote

$$\mathbb{E}C_x = \mathbb{E}\sum_{y \in S} I_y = \sum_{y \in S} \mathbb{E}I_y.$$

Question: where did we use independence?

Answer

Nowhere. Linearity of expectation does not require the indicators I_y to be independent.

The only probabilistic fact used for each fixed $y \neq x$ is

$$\mathbb{P}(h(y) = h(x)) \leq \frac{1}{m}.$$

This suggests a useful strategy: examine the proof and weaken the randomness assumption to what the proof requires.

Universal hashing

A family $\mathcal{H} \subseteq \{h : U \to [m]\}$ is **universal** if, for every fixed distinct pair $x, y \in U$,

$$\mathbb{P}_{h \sim \mathcal{H}}(h(x) = h(y)) \leq \frac{1}{m}.$$

The probability is over the random seed choosing h. The keys are fixed first.

With this condition, the chained-lookup proof is unchanged:

$$\mathbb{E}C_x = \sum_{y \in S} \mathbb{P}(h(y) = h(x)) \leq \frac{n}{m}.$$

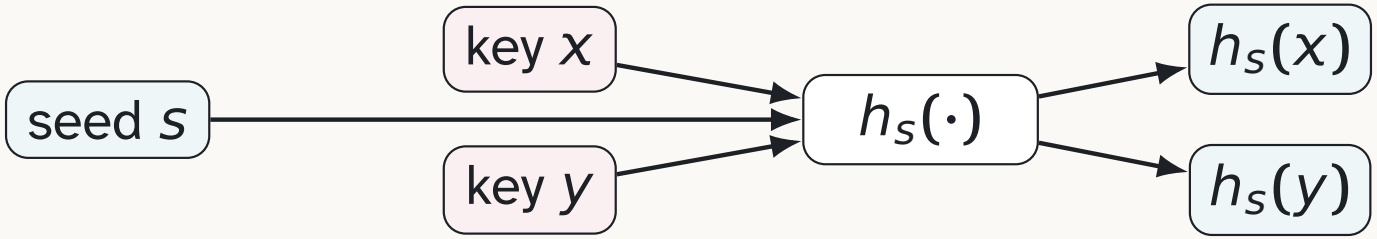
This condition is what the preceding expectation bound requires.

A hash function is chosen once

A hash family is a collection

$$\mathcal{H} = \{h_s : s \in \Sigma\}.$$

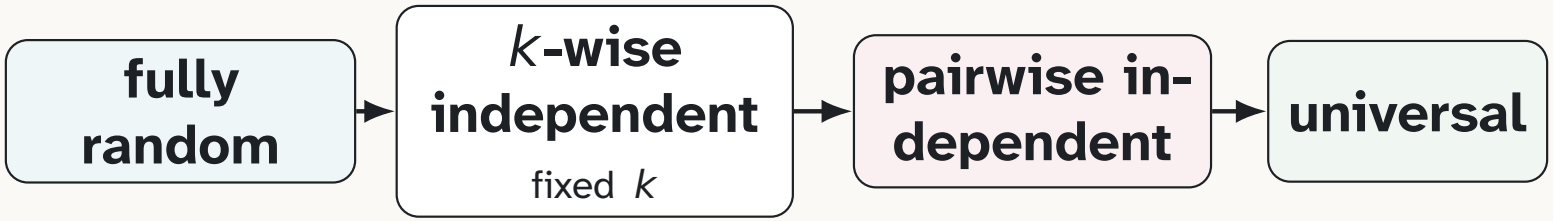
To build the table, sample one seed s, store it, and then use the deterministic function h_s for every insertion and query.



There is no new randomness at query time. The randomness is chosen once when the data structure is initialized.

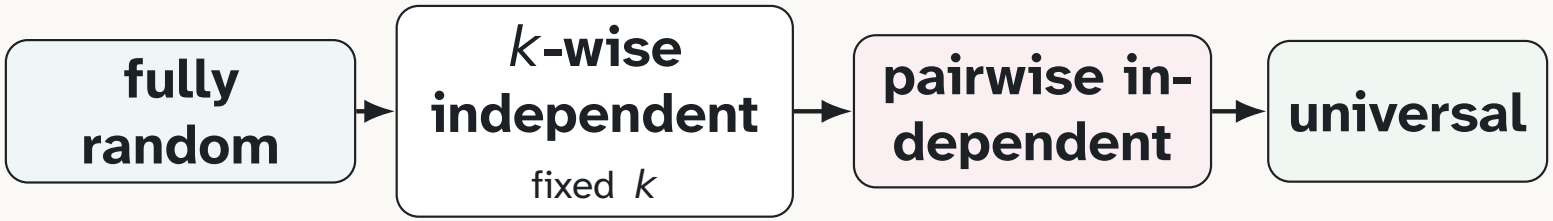
A hierarchy of randomness guarantees

These notions are related by implication. Fix some $k \geq 2$.



A hierarchy of randomness guarantees

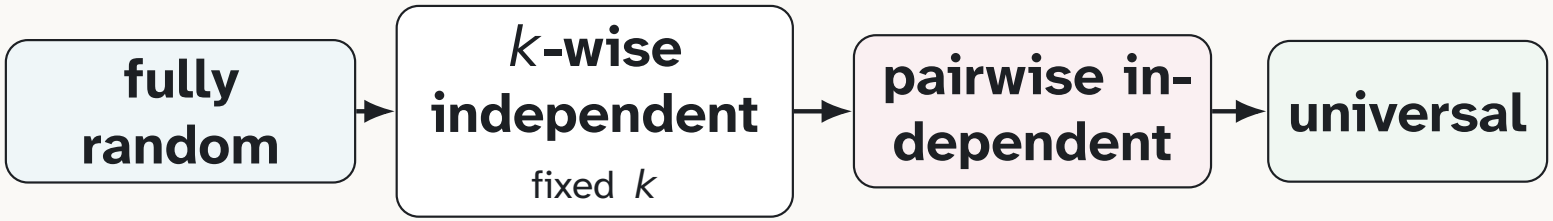
These notions are related by implication. Fix some $k \geq 2$.



- **Fully random:** the whole vector $(h(x))_{x \in U}$ is mutually independent and uniform.

A hierarchy of randomness guarantees

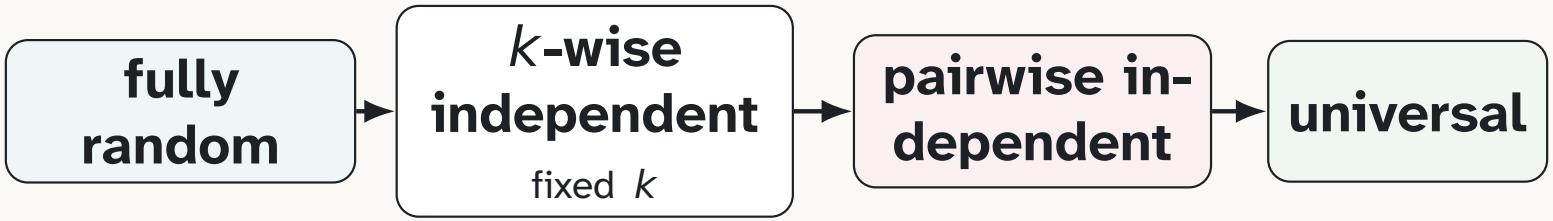
These notions are related by implication. Fix some $k \geq 2$.



- **Fully random:** the whole vector $(h(x))_{x \in U}$ is mutually independent and uniform.
- **k-wise independent:** every fixed k-tuple of distinct inputs has independent uniform hash values.

A hierarchy of randomness guarantees

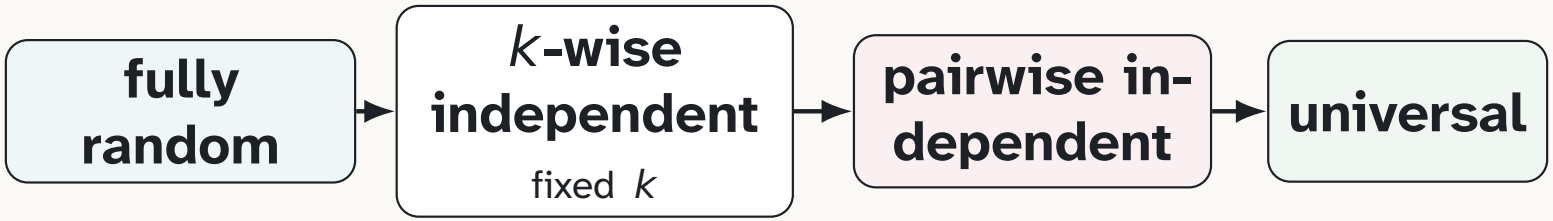
These notions are related by implication. Fix some $k \geq 2$.



- **Fully random:** the whole vector $(h(x))_{x \in U}$ is mutually independent and uniform.
- **k-wise independent:** every fixed k-tuple of distinct inputs has independent uniform hash values.
- **Universal:** for fixed distinct x, y, we only control the collision event $h(x) = h(y)$.

A hierarchy of randomness guarantees

These notions are related by implication. Fix some $k \geq 2$.



- **Fully random:** the whole vector $(h(x))_{x \in U}$ is mutually independent and uniform.
- **k-wise independent:** every fixed k-tuple of distinct inputs has independent uniform hash values.
- **Universal:** for fixed distinct x, y, we only control the collision event $h(x) = h(y)$.

Chained lookup needs only the weakest property shown here.

First construction: pairwise independence

We have found the condition needed for chaining:

$$\Pr(h(x) = h(y)) \leq \frac{1}{m} \qquad \text{for every fixed } x \neq y.$$

First construction: pairwise independence

We have found the condition needed for chaining:

$$\Pr(h(x) = h(y)) \leq \frac{1}{m} \qquad \text{for every fixed } x \neq y.$$

A stronger condition is that, for every fixed pair $x \neq y$, the pair of outputs

$$(h(x), h(y))$$

is uniform on all m^2 ordered bucket pairs.

First construction: pairwise independence

We have found the condition needed for chaining:

$$\Pr(h(x) = h(y)) \leq \frac{1}{m} \qquad \text{for every fixed } x \neq y.$$

A stronger condition is that, for every fixed pair $x \neq y$, the pair of outputs

$$(h(x), h(y))$$

is uniform on all m^2 ordered bucket pairs.
Equivalently, after choosing the seed uniformly, every desired pair of outputs

$$h(x) = u, \qquad h(y) = v$$

should be realized by the same number of seeds.

First construction: pairwise independence

We have found the condition needed for chaining:

$$\Pr(h(x) = h(y)) \leq \frac{1}{m} \qquad \text{for every fixed } x \neq y.$$

A stronger condition is that, for every fixed pair $x \neq y$, the pair of outputs

$$(h(x), h(y))$$

is uniform on all m^2 ordered bucket pairs.
Equivalently, after choosing the seed uniformly, every desired pair of outputs

$$h(x) = u, \qquad h(y) = v$$

should be realized by the same number of seeds.
This suggests a concrete design problem: can a small seed force two prescribed output values?

Trying a two-parameter family

Try the two-parameter family

$$h_{a,b}(z) = az + b.$$

Trying a two-parameter family

Try the two-parameter family

$$h_{a,b}(z) = az + b.$$

For fixed distinct inputs x, y, and prescribed outputs u, v, the constraints are

$$ax + b = u, \qquad ay + b = v.$$

Trying a two-parameter family

Try the two-parameter family

$$h_{a,b}(z) = az + b.$$

For fixed distinct inputs x, y, and prescribed outputs u, v, the constraints are

$$ax + b = u, \qquad ay + b = v.$$

If these two equations determine exactly one seed (a, b), then every output pair (u, v) occurs equally often.

Trying a two-parameter family

Try the two-parameter family

$$h_{a,b}(z) = az + b.$$

For fixed distinct inputs x, y, and prescribed outputs u, v, the constraints are

$$ax + b = u, \qquad ay + b = v.$$

If these two equations determine exactly one seed (a, b), then every output pair (u, v) occurs equally often.
That would imply

$$(h(x), h(y)) \text{ is uniform,}$$

and hence the family is pairwise independent.

An affine example over $\mathbb{F}_7$

Work in $\mathbb{F}_7$. Suppose we want

$$h(2) = 4, \qquad h(5) = 1,$$

for $h(x) = ax + b \pmod 7$.

An affine example over $\mathbb{F}_7$

Work in $\mathbb{F}_7$. Suppose we want

$$h(2) = 4, \qquad h(5) = 1,$$

for $h(x) = ax + b \pmod 7$.
Then

$$2a + b = 4, \qquad 5a + b = 1.$$

An affine example over $\mathbb{F}_7$

Work in $\mathbb{F}_7$. Suppose we want

$$h(2) = 4, \qquad h(5) = 1,$$

for $h(x) = ax + b \pmod 7$.
Then

$$2a + b = 4, \qquad 5a + b = 1.$$

Subtracting gives

$$3a = 4 \pmod 7.$$

An affine example over $\mathbb{F}_7$

Work in $\mathbb{F}_7$. Suppose we want

$$h(2) = 4, \qquad h(5) = 1,$$

for $h(x) = ax + b \pmod 7$.
Then

$$2a + b = 4, \qquad 5a + b = 1.$$

Subtracting gives

$$3a = 4 \pmod 7.$$

Since $3^{-1} = 5 \pmod 7$,

$$a = 4 \cdot 5 = 6 \pmod 7.$$

An affine example over $\mathbb{F}_7$

Work in $\mathbb{F}_7$. Suppose we want

$$h(2) = 4, \qquad h(5) = 1,$$

for $h(x) = ax + b \pmod 7$.
Then

$$2a + b = 4, \qquad 5a + b = 1.$$

Subtracting gives

$$3a = 4 \pmod 7.$$

Since $3^{-1} = 5 \pmod 7$,

$$a = 4 \cdot 5 = 6 \pmod 7.$$

Then

$$b = 4 - 2a = 6 \pmod 7.$$

An affine example over $\mathbb{F}_7$

Work in $\mathbb{F}_7$. Suppose we want

$$h(2) = 4, \qquad h(5) = 1,$$

for $h(x) = ax + b \pmod 7$.
Then

$$2a + b = 4, \qquad 5a + b = 1.$$

Subtracting gives

$$3a = 4 \pmod 7.$$

Since $3^{-1} = 5 \pmod 7$,

$$a = 4 \cdot 5 = 6 \pmod 7.$$

Then

$$b = 4 - 2a = 6 \pmod 7.$$

For this fixed pair of inputs, this prescribed pair of outputs determines exactly one seed (a, b). We now prove the same fact for every $x \neq y$ and every $u, v \in \mathbb{F}_p$.

Prime fields

Let p be prime, and write

$$\mathbb{F}_p = \{0, 1, \ldots, p-1\}$$

with arithmetic modulo p.

The property we use is

$$z \neq 0 \quad \Longrightarrow \quad z^{-1} \text{ exists in } \mathbb{F}_p.$$

Equivalently, if $z \neq 0$, the equation

$$az = c$$

has the unique solution $a = cz^{-1}$.

This fails for composite moduli. Modulo 6,

$$2 \cdot 3 \equiv 0 \pmod 6,$$

so a nonzero residue need not be invertible.

Pairwise independence of affine hashing

Choose $a, b \in \mathbb{F}_p$ uniformly and independently, and define

$$h_{a,b}(x) = ax + b \pmod{p}.$$

Fix distinct $x, y \in \mathbb{F}_p$. To control collisions, we first determine the distribution of

$$(h(x), h(y)).$$

Pairwise independence of affine hashing

Choose $a, b \in \mathbb{F}_p$ uniformly and independently, and define

$$h_{a,b}(x) = ax + b \pmod{p}.$$

Fix distinct $x, y \in \mathbb{F}_p$. To control collisions, we first determine the distribution of

$$(h(x), h(y)).$$

For any target outputs $u, v \in \mathbb{F}_p$, solve

$$ax + b = u, \qquad ay + b = v.$$

Pairwise independence of affine hashing

Choose $a, b \in \mathbb{F}_p$ uniformly and independently, and define

$$h_{a,b}(x) = ax + b \pmod{p}.$$

Fix distinct $x, y \in \mathbb{F}_p$. To control collisions, we first determine the distribution of

$$(h(x), h(y)).$$

For any target outputs $u, v \in \mathbb{F}_p$, solve

$$ax + b = u, \qquad ay + b = v.$$

Subtract:

$$a(x - y) = u - v.$$

Pairwise independence of affine hashing

Choose $a, b \in \mathbb{F}_p$ uniformly and independently, and define

$$h_{a,b}(x) = ax + b \pmod{p}.$$

Fix distinct $x, y \in \mathbb{F}_p$. To control collisions, we first determine the distribution of

$$(h(x), h(y)).$$

For any target outputs $u, v \in \mathbb{F}_p$, solve

$$ax + b = u, \qquad ay + b = v.$$

Subtract:

$$a(x - y) = u - v.$$

Since $x - y \neq 0$, division is legal:

$$a = (u - v)(x - y)^{-1}, \qquad b = u - ax.$$

Pairwise independence of affine hashing

Choose $a, b \in \mathbb{F}_p$ uniformly and independently, and define

$$h_{a,b}(x) = ax + b \pmod{p}.$$

Fix distinct $x, y \in \mathbb{F}_p$. To control collisions, we first determine the distribution of

$$(h(x), h(y)).$$

For any target outputs $u, v \in \mathbb{F}_p$, solve

$$ax + b = u, \qquad ay + b = v.$$

Subtract:

$$a(x - y) = u - v.$$

Since $x - y \neq 0$, division is legal:

$$a = (u - v)(x - y)^{-1}, \qquad b = u - ax.$$

Thus exactly one of the p^2 seeds sends (x, y) to (u, v).

Affine hashing is universal

Because each output pair $(u, v) \in \mathbb{F}_p^2$ is produced by exactly one of the p^2 seeds,

$$(h(x), h(y)) \text{ is uniform on } \mathbb{F}_p^2.$$

So for fixed $x \neq y$, the two outputs are independent and uniform.

Affine hashing is universal

Because each output pair $(u, v) \in \mathbb{F}_p^2$ is produced by exactly one of the p^2 seeds,

$$(h(x), h(y)) \text{ is uniform on } \mathbb{F}_p^2.$$

So for fixed $x \neq y$, the two outputs are independent and uniform.
Now compute the collision probability explicitly:

$$\Pr(h(x) = h(y)) = \sum_{u \in \mathbb{F}_p} \Pr(h(x) = u) \Pr(h(y) = u \mid h(x) = u)$$

$$= \sum_{u \in \mathbb{F}_p} \frac{1}{p} \cdot \frac{1}{p} = \frac{1}{p}.$$

Affine hashing is universal

Because each output pair $(u, v) \in \mathbb{F}_p^2$ is produced by exactly one of the p^2 seeds,

$$(h(x), h(y)) \text{ is uniform on } \mathbb{F}_p^2.$$

So for fixed $x \neq y$, the two outputs are independent and uniform.
Now compute the collision probability explicitly:

$$\Pr(h(x) = h(y)) = \sum_{u \in \mathbb{F}_p} \Pr(h(x) = u) \Pr(h(y) = u \mid h(x) = u)$$

$$= \sum_{u \in \mathbb{F}_p} \frac{1}{p} \cdot \frac{1}{p} = \frac{1}{p}.$$

Therefore affine hashing is universal. The seed is only two field elements.

Can we hash using only bit operations?

We have shown that full randomness is unnecessary: a short affine seed already preserves the fixed-pair collision bound.

Can we hash using only bit operations?

We have shown that full randomness is unnecessary: a short affine seed already preserves the fixed-pair collision bound.
But arithmetic over $\mathbb{F}_p$ may require:

- choosing a prime field large enough for the encoded universe;
- interpreting machine words as field elements;
- multiplication and modular reduction.

Can we hash using only bit operations?

We have shown that full randomness is unnecessary: a short affine seed already preserves the fixed-pair collision bound.
But arithmetic over $\mathbb{F}_p$ may require:

- choosing a prime field large enough for the encoded universe;
- interpreting machine words as field elements;
- multiplication and modular reduction.

Can we preserve universality using only operations on the bits of the key?

Can we hash using only bit operations?

We have shown that full randomness is unnecessary: a short affine seed already preserves the fixed-pair collision bound.
But arithmetic over $\mathbb{F}_p$ may require:

- choosing a prime field large enough for the encoded universe;
- interpreting machine words as field elements;
- multiplication and modular reduction.

Can we preserve universality using only operations on the bits of the key? Assume keys are w-bit strings and $m = 2^\ell$ buckets. Then a hash value is naturally an ℓ-bit bucket label:

$$\{0,1\}^w \longrightarrow \{0,1\}^\ell.$$

Arithmetic over bits

Over $\mathbb{F}_2 = \{0, 1\}$, addition is xor and multiplication is and:

$\oplus$	0	1
0	0	1
1	1	0

$\wedge$	0	1
0	0	0
1	0	1

Arithmetic over bits

Over $\mathbb{F}_2 = \{0, 1\}$, addition is xor and multiplication is and:

$\oplus$	0	1
0	0	1
1	1	0

$\wedge$	0	1
0	0	0
1	0	1

For $a, x \in \mathbb{F}_2^w$, define

$$a^\top x = \bigoplus_{j=0}^{w-1} (a_j \wedge x_j).$$

Arithmetic over bits

Over $\mathbb{F}_2 = \{0, 1\}$, addition is xor and multiplication is and:

$\oplus$	0	1
0	0	1
1	1	0

$\wedge$	0	1
0	0	0
1	0	1

For $a, x \in \mathbb{F}_2^w$, define

$$a^\top x = \bigoplus_{j=0}^{w-1} (a_j \wedge x_j).$$

For example, if $a = (1, 1, 0, 1)$ and $x = (1, 0, 1, 1)$, then

$$\begin{aligned} a^\top x &= (1 \wedge 1) \oplus (1 \wedge 0) \oplus (0 \wedge 1) \oplus (1 \wedge 1) \\ &= 1 \oplus 0 \oplus 0 \oplus 1 \\ &= 0. \end{aligned}$$

Random binary matrices

Let $m = 2^\ell$. Choose a random matrix

$$A \in \mathbb{F}_2^{\ell \times w}$$

with independent fair bits, and define

$$h_A(x) = Ax \in \mathbb{F}_2^\ell.$$

$$A = \begin{bmatrix} a_0^\top \\ a_1^\top \\ \vdots \\ a_{\ell-1}^\top \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{w-1} \end{bmatrix} = \begin{bmatrix} a_0^\top x \\ a_1^\top x \\ \vdots \\ a_{\ell-1}^\top x \end{bmatrix}$$

The output is an ℓ-bit bucket label, computed using only AND/XOR operations.

Bounding the collision probability

Fix distinct keys $x, y \in \mathbb{F}_2^w$. We want to show

$$\Pr_A(h_A(x) = h_A(y)) \leq 2^{-\ell}.$$

Bounding the collision probability

Fix distinct keys $x, y \in \mathbb{F}_2^w$. We want to show

$$\Pr_A(h_A(x) = h_A(y)) \leq 2^{-\ell}.$$

Start from the collision event:

$$h_A(x) = h_A(y) \quad \Longleftrightarrow \quad Ax = Ay.$$

Bounding the collision probability

Fix distinct keys $x, y \in \mathbb{F}_2^w$. We want to show

$$\Pr_A(h_A(x) = h_A(y)) \leq 2^{-\ell}.$$

Start from the collision event:

$$h_A(x) = h_A(y) \quad \Longleftrightarrow \quad Ax = Ay.$$

Move all terms to one side:

$$A(x - y) = 0.$$

Over $\mathbb{F}_2$, subtraction equals addition, so

$$d = x - y = x \oplus y \neq 0.$$

Bounding the collision probability

Fix distinct keys $x, y \in \mathbb{F}_2^w$. We want to show

$$\Pr_A(h_A(x) = h_A(y)) \leq 2^{-\ell}.$$

Start from the collision event:

$$h_A(x) = h_A(y) \quad \Longleftrightarrow \quad Ax = Ay.$$

Move all terms to one side:

$$A(x - y) = 0.$$

Over $\mathbb{F}_2$, subtraction equals addition, so

$$d = x - y = x \oplus y \neq 0.$$

Thus

$$h_A(x) = h_A(y) \quad \Longleftrightarrow \quad Ad = 0$$

for a fixed nonzero vector d.

One random row

The event $Ad = 0$ means that every row a_r of A satisfies

$$a_r^\top d = 0.$$

First compute this probability for one random row a.

One random row

The event $Ad = 0$ means that every row a_r of A satisfies

$$a_r^\top d = 0.$$

First compute this probability for one random row a.
Choose an index k with $d_k = 1$, and condition on all row bits except a_k.
Let

$$C = \bigoplus_{j \neq k} (a_j \wedge d_j).$$

After conditioning, C is fixed and

$$a^\top d = a_k \oplus C.$$

One random row

The event $Ad = 0$ means that every row a_r of A satisfies

$$a_r^\top d = 0.$$

First compute this probability for one random row a.
Choose an index k with $d_k = 1$, and condition on all row bits except a_k.
Let

$$C = \bigoplus_{j \neq k} (a_j \wedge d_j).$$

After conditioning, C is fixed and

$$a^\top d = a_k \oplus C.$$

Therefore

$$\Pr(a^\top d = 0 \mid (a_j)_{j \neq k}) = \Pr(a_k = C) = \frac{1}{2}.$$

One random row

The event $Ad = 0$ means that every row a_r of A satisfies

$$a_r^\top d = 0.$$

First compute this probability for one random row a.
Choose an index k with $d_k = 1$, and condition on all row bits except a_k.
Let

$$C = \bigoplus_{j \neq k} (a_j \wedge d_j).$$

After conditioning, C is fixed and

$$a^\top d = a_k \oplus C.$$

Therefore

$$\Pr(a^\top d = 0 \mid (a_j)_{j \neq k}) = \Pr(a_k = C) = \frac{1}{2}.$$

Taking expectation over the conditioned bits gives $\Pr(a^\top d = 0) = 1/2$.
Since the rows are independent,

$$\Pr(Ad = 0) = \prod_{r=0}^{\ell-1} \Pr(a_r^\top d = 0) = 2^{-\ell} = \frac{1}{m}.$$

Random bits saved so far

For w-bit keys, $|U| = 2^w$, and $m = 2^\ell$.

family	random seed
fully random table	$2^w \ell$ bits
random binary matrix	$w\ell$ bits

Random bits saved so far

For w-bit keys, $|U| = 2^w$, and $m = 2^\ell$.

family	random seed
fully random table	$2^w \ell$ bits
random binary matrix	$w\ell$ bits

For 64-bit keys and 2^{20} buckets:

$$\begin{aligned} \text{fully random table} &: 2^{64} \cdot 20 \text{ bits,} \\ \text{random binary matrix} &: 64 \cdot 20 = 1280 \text{ bits} = 160 \text{ bytes.} \end{aligned}$$

Random bits saved so far

For w-bit keys, $|U| = 2^w$, and $m = 2^\ell$.

family	random seed
fully random table	$2^w \ell$ bits
random binary matrix	$w\ell$ bits

For 64-bit keys and 2^{20} buckets:

$$\begin{aligned} \text{fully random table} &: 2^{64} \cdot 20 \text{ bits,} \\ \text{random binary matrix} &: 64 \cdot 20 = 1280 \text{ bits} = 160 \text{ bytes.} \end{aligned}$$

The matrix construction is already a huge reduction. Next question: can we reduce the $w\ell$-bit seed while preserving universality?

Toeplitz matrices reuse diagonals

Choose random bits

$$s_0, s_1, \ldots, s_{w+\ell-2}$$

and define

$$A_{rj} = s_{r-j+w-1}, \qquad 0 \le r \le \ell - 1, \quad 0 \le j \le w - 1.$$

Entries on the same diagonal share the same random bit.

$$A = \begin{array}{|c|c|c|c|c|c|} \hline s_5 & s_4 & s_3 & s_2 & s_1 & s_0 \\ \hline s_6 & s_5 & s_4 & s_3 & s_2 & s_1 \\ \hline s_7 & s_6 & s_5 & s_4 & s_3 & s_2 \\ \hline s_8 & s_7 & s_6 & s_5 & s_4 & s_3 \\ \hline \end{array}$$

seed length $= w + \ell - 1$ bits instead of $w\ell$.

Toeplitz matrices reuse diagonals

Choose random bits

$$s_0, s_1, \ldots, s_{w+\ell-2}$$

and define

$$A_{rj} = s_{r-j+w-1}, \qquad 0 \le r \le \ell - 1, \quad 0 \le j \le w - 1.$$

Entries on the same diagonal share the same random bit.

$$A = \begin{array}{|c|c|c|c|c|c|} \hline s_5 & s_4 & s_3 & s_2 & s_1 & s_0 \\ \hline s_6 & s_5 & s_4 & s_3 & s_2 & s_1 \\ \hline s_7 & s_6 & s_5 & s_4 & s_3 & s_2 \\ \hline s_8 & s_7 & s_6 & s_5 & s_4 & s_3 \\ \hline \end{array}$$

seed length $= w + \ell - 1$ bits instead of $w\ell$.

Goal: prove

$$\Pr_A(Ax = Ay) = 2^{-\ell} = \frac{1}{m}$$

for every fixed distinct x, y, or equivalently $\Pr_A(Ad = 0) = 2^{-\ell}$ for every fixed $d \neq 0$.

A one-coordinate example

Use $\ell = 3$, $w = 5$:

$$A = \begin{bmatrix} s_4 & s_3 & s_2 & s_1 & s_0 \\ s_5 & s_4 & s_3 & s_2 & s_1 \\ s_6 & s_5 & s_4 & s_3 & s_2 \end{bmatrix}.$$

A one-coordinate example

Use $\ell = 3$, $w = 5$:

$$A = \begin{bmatrix} s_4 & s_3 & s_2 & s_1 & s_0 \\ s_5 & s_4 & s_3 & s_2 & s_1 \\ s_6 & s_5 & s_4 & s_3 & s_2 \end{bmatrix}.$$

Take

$$d = e_4 = (0, 0, 0, 0, 1)^{\mathsf{T}}.$$

Then

$$Ad = \begin{bmatrix} s_0 \\ s_1 \\ s_2 \end{bmatrix}.$$

A one-coordinate example

Use $\ell = 3$, $w = 5$:

$$A = \begin{bmatrix} s_4 & s_3 & s_2 & s_1 & s_0 \\ s_5 & s_4 & s_3 & s_2 & s_1 \\ s_6 & s_5 & s_4 & s_3 & s_2 \end{bmatrix}.$$

Take

$$d = e_4 = (0, 0, 0, 0, 1)^{\mathsf{T}}.$$

Then

$$Ad = \begin{bmatrix} s_0 \\ s_1 \\ s_2 \end{bmatrix}.$$

Even though d has only one nonzero coordinate, that coordinate meets a different seed bit in each output row. Hence

$$\Pr(Ad = 0) = \Pr(s_0 = s_1 = s_2 = 0) = 2^{-3}.$$

A dependent Toeplitz example

Again take $\ell = 3$, $w = 5$, and

$$A = \begin{bmatrix} s_4 & s_3 & s_2 & s_1 & s_0 \\ s_5 & s_4 & s_3 & s_2 & s_1 \\ s_6 & s_5 & s_4 & s_3 & s_2 \end{bmatrix}.$$

A dependent Toeplitz example

Again take $\ell = 3$, $w = 5$, and

$$A = \begin{bmatrix} s_4 & s_3 & s_2 & s_1 & s_0 \\ s_5 & s_4 & s_3 & s_2 & s_1 \\ s_6 & s_5 & s_4 & s_3 & s_2 \end{bmatrix}.$$

For

$$d = (0, 1, 0, 1, 0)^{\mathsf{T}},$$

we get

$$Ad = \begin{bmatrix} s_3 \oplus s_1 \\ s_4 \oplus s_2 \\ s_5 \oplus s_3 \end{bmatrix}.$$

A dependent Toeplitz example

Again take $\ell = 3$, $w = 5$, and

$$A = \begin{bmatrix} s_4 & s_3 & s_2 & s_1 & s_0 \\ s_5 & s_4 & s_3 & s_2 & s_1 \\ s_6 & s_5 & s_4 & s_3 & s_2 \end{bmatrix}.$$

For

$$d = (0, 1, 0, 1, 0)^{\mathsf{T}},$$

we get

$$Ad = \begin{bmatrix} s_3 \oplus s_1 \\ s_4 \oplus s_2 \\ s_5 \oplus s_3 \end{bmatrix}.$$

The output bits are not independent: s_3 appears in rows 0 and 2. So the independent-row proof for a fully random matrix does not apply.

Follow the column of the first 1

For

$$d = (0, 1, 0, 1, 0)^{\mathsf{T}},$$

the first nonzero coordinate is $k = 1$. Since $d_1 = 1$, every row of Ad contains the entry in column 1 of A.

Follow the column of the first 1

For

$$d = (0, 1, 0, 1, 0)^{\mathsf{T}},$$

the first nonzero coordinate is $k = 1$. Since $d_1 = 1$, every row of Ad contains the entry in column 1 of A.

$$A = \begin{bmatrix} s_4 & \textcircled{s_3} & s_2 & s_1 & s_0 \\ s_5 & \textcircled{s_4} & s_3 & s_2 & s_1 \\ s_6 & \textcircled{s_5} & s_4 & s_3 & s_2 \end{bmatrix}$$

Follow the column of the first 1

For

$$d = (0, 1, 0, 1, 0)^{\mathsf{T}},$$

the first nonzero coordinate is $k = 1$. Since $d_1 = 1$, every row of Ad contains the entry in column 1 of A.

$$A = \begin{bmatrix} s_4 & \textcircled{s_3} & s_2 & s_1 & s_0 \\ s_5 & \textcircled{s_4} & s_3 & s_2 & s_1 \\ s_6 & \textcircled{s_5} & s_4 & s_3 & s_2 \end{bmatrix}$$

The highlighted column contributes one seed bit to each output row:

$$Ad = \begin{bmatrix} \textcircled{s_3} \oplus s_1 \\ \textcircled{s_4} \oplus s_2 \\ \textcircled{s_5} \oplus s_3 \end{bmatrix}.$$

The repeated s_3 is why the independent-row proof does not apply directly.

Sequential conditioning in the example

Condition first on all seed bits outside the highlighted column. In this example that fixes s_1 and s_2.

Sequential conditioning in the example

Condition first on all seed bits outside the highlighted column. In this example that fixes s_1 and s_2.
Then

$$(Ad)_0 = s_3 \oplus \text{known},$$
$$(Ad)_1 = s_4 \oplus \text{known},$$
$$(Ad)_2 = s_5 \oplus s_3.$$

Sequential conditioning in the example

Condition first on all seed bits outside the highlighted column. In this example that fixes s_1 and s_2.
Then

$$(Ad)_0 = s_3 \oplus \text{known},$$
$$(Ad)_1 = s_4 \oplus \text{known},$$
$$(Ad)_2 = s_5 \oplus s_3.$$

Process the rows from top to bottom:

$$\Pr((Ad)_0 = 0 \mid s_1, s_2) = \frac{1}{2},$$
$$\Pr((Ad)_1 = 0 \mid (Ad)_0 = 0, s_1, s_2) = \frac{1}{2},$$
$$\Pr((Ad)_2 = 0 \mid (Ad)_0 = (Ad)_1 = 0, s_1, s_2) = \frac{1}{2}.$$

Sequential conditioning in the example

Condition first on all seed bits outside the highlighted column. In this example that fixes s_1 and s_2.
Then

$$(Ad)_0 = s_3 \oplus \text{known},$$
$$(Ad)_1 = s_4 \oplus \text{known},$$
$$(Ad)_2 = s_5 \oplus s_3.$$

Process the rows from top to bottom:

$$\Pr((Ad)_0 = 0 \mid s_1, s_2) = \frac{1}{2},$$
$$\Pr((Ad)_1 = 0 \mid (Ad)_0 = 0, s_1, s_2) = \frac{1}{2},$$
$$\Pr((Ad)_2 = 0 \mid (Ad)_0 = (Ad)_1 = 0, s_1, s_2) = \frac{1}{2}.$$

So $\Pr(Ad = 0 \mid s_1, s_2) = 2^{-3}$, and therefore $\Pr(Ad = 0) = 2^{-3}$.

The first nonzero coordinate selects a column

Fix $d \neq 0$, and let

$$k = \min\{j : d_j = 1\}.$$

Because $d_k = 1$, row r of Ad contains the matrix entry in row r, column k:

$$A_{rk} = s_{r-k+w-1}.$$

The first nonzero coordinate selects a column

Fix $d \neq 0$, and let

$$k = \min\{j : d_j = 1\}.$$

Because $d_k = 1$, row r of Ad contains the matrix entry in row r, column k:

$$A_{rk} = s_{r-k+w-1}.$$

Thus the column selected by the first 1 of d is

$$A_{0k}, A_{1k}, \ldots, A_{\ell-1,k} = s_{w-1-k}, s_{w-k}, \ldots, s_{w+\ell-2-k}.$$

These are ℓ distinct independent seed bits.

The first nonzero coordinate selects a column

Fix $d \neq 0$, and let

$$k = \min\{j : d_j = 1\}.$$

Because $d_k = 1$, row r of Ad contains the matrix entry in row r, column k:

$$A_{rk} = s_{r-k+w-1}.$$

Thus the column selected by the first 1 of d is

$$A_{0k}, A_{1k}, \ldots, A_{\ell-1,k} = s_{w-1-k}, s_{w-k}, \ldots, s_{w+\ell-2-k}.$$

These are ℓ distinct independent seed bits.
Condition on every seed bit outside this column. The entries in column k remain independent fair bits.

General Toeplitz proof

Since $d_j = 0$ for $j < k$, row r satisfies

$$\begin{aligned}(Ad)_r &= \bigoplus_{j=0}^{w-1} (A_{rj} \wedge d_j) \\ &= A_{rk} \oplus \bigoplus_{j>k} (A_{rj} \wedge d_j).\end{aligned}$$

General Toeplitz proof

Since $d_j = 0$ for $j < k$, row r satisfies

$$(Ad)_r = \bigoplus_{j=0}^{w-1} (A_{rj} \wedge d_j)$$
$$= A_{rk} \oplus \bigoplus_{j>k} (A_{rj} \wedge d_j).$$

For $j > k$,

$$A_{rj} = s_{r-j+w-1} \quad \text{has seed index} \quad r - j + w - 1 < r - k + w - 1.$$

So any unfixed column-k seed bit that appears in the remaining sum must come from an earlier row.

General Toeplitz proof

Since $d_j = 0$ for $j < k$, row r satisfies

$$(Ad)_r = \bigoplus_{j=0}^{w-1} (A_{rj} \wedge d_j)$$
$$= A_{rk} \oplus \bigoplus_{j>k} (A_{rj} \wedge d_j).$$

For $j > k$,

$$A_{rj} = s_{r-j+w-1} \quad \text{has seed index} \quad r - j + w - 1 < r - k + w - 1.$$

So any unfixed column-k seed bit that appears in the remaining sum must come from an earlier row.
After conditioning on all seed bits outside column k,

$$(Ad)_r = A_{rk} \oplus F_r(A_{0k}, \ldots, A_{r-1,k}).$$

Toeplitz universality

Process the rows in order. At row r, the bit A_{rk} has not appeared in earlier rows, and it is still an independent fair bit.

Toeplitz universality

Process the rows in order. At row r, the bit A_{rk} has not appeared in earlier rows, and it is still an independent fair bit.
Therefore

$$\Pr\big((Ad)_r = 0 \mid (Ad)_0 = \cdots = (Ad)_{r-1} = 0, \text{ all seed bits outside column } k\big) = \frac{1}{2}.$$

Toeplitz universality

Process the rows in order. At row r, the bit A_{rk} has not appeared in earlier rows, and it is still an independent fair bit.
Therefore

$$\Pr\big((Ad)_r = 0 \mid (Ad)_0 = \cdots = (Ad)_{r-1} = 0, \text{ all seed bits outside column } k\big) = \frac{1}{2}.$$

By the chain rule,

$$\Pr(Ad = 0 \mid \text{all seed bits outside column } k) = 2^{-\ell}.$$

Averaging over the conditioned bits gives

$$\Pr(Ad = 0) = 2^{-\ell} = \frac{1}{m}.$$

Seed lengths

For w-bit keys and $m = 2^\ell$ buckets:

family	seed size	operations
fully random table	$2^w \ell$ bits	table lookup
random binary matrix	$w\ell$ bits	AND/XOR
Toeplitz matrix	$w + \ell - 1$ bits	AND/XOR with reused bits

Seed lengths

For w-bit keys and $m = 2^\ell$ buckets:

family	seed size	operations
fully random table	$2^w \ell$ bits	table lookup
random binary matrix	$w\ell$ bits	AND/XOR
Toeplitz matrix	$w + \ell - 1$ bits	AND/XOR with reused bits

For 64-bit keys and 2^{20} buckets:

$$\begin{aligned} \text{fully random} &: 2^{64} \cdot 20 \text{ bits}, \\ \text{binary matrix} &: 1280 \text{ bits}, \\ \text{Toeplitz} &: 64 + 20 - 1 = 83 \text{ bits}. \end{aligned}$$

Seed lengths

For w-bit keys and $m = 2^\ell$ buckets:

family	seed size	operations
fully random table	$2^w \ell$ bits	table lookup
random binary matrix	$w\ell$ bits	AND/XOR
Toeplitz matrix	$w + \ell - 1$ bits	AND/XOR with reused bits

For 64-bit keys and 2^{20} buckets:

$$\begin{aligned} \text{fully random} &: 2^{64} \cdot 20 \text{ bits}, \\ \text{binary matrix} &: 1280 \text{ bits}, \\ \text{Toeplitz} &: 64 + 20 - 1 = 83 \text{ bits}. \end{aligned}$$

All three preserve the chaining-relevant guarantee

$$\Pr(h(x) = h(y)) \le \frac{1}{m}$$

for each fixed distinct pair.

Earlier hash values change the state

For chained hashing, every probability we used concerned a pair of keys fixed before the hash function was sampled.

Earlier hash values change the state

For chained hashing, every probability we used concerned a pair of keys fixed before the hash function was sampled.
In the power-of-two-choices process from Lecture 6, the state at time t was produced by all previous placements.

Earlier hash values change the state

For chained hashing, every probability we used concerned a pair of keys fixed before the hash function was sampled.
In the power-of-two-choices process from Lecture 6, the state at time t was produced by all previous placements.
For example, the set of heavily loaded bins at time t is itself determined by earlier hash values.

Earlier hash values change the state

For chained hashing, every probability we used concerned a pair of keys fixed before the hash function was sampled.
In the power-of-two-choices process from Lecture 6, the state at time t was produced by all previous placements.
For example, the set of heavily loaded bins at time t is itself determined by earlier hash values.
Before returning to two choices, let us see concretely why pairwise independence need not survive conditioning.

What happens after we observe two hash values?

Let

$$h(x) = ax + b \pmod{p}.$$

Before seeing any outputs, every fixed pair $(h(x), h(y))$ for $x \neq y$ is uniform.

What happens after we observe two hash values?

Let

$$h(x) = ax + b \pmod{p}.$$

Before seeing any outputs, every fixed pair $(h(x), h(y))$ for $x \neq y$ is uniform.
But suppose we learn

$$h(0) = u, \qquad h(1) = v.$$

What happens after we observe two hash values?

Let

$$h(x) = ax + b \pmod{p}.$$

Before seeing any outputs, every fixed pair $(h(x), h(y))$ for $x \neq y$ is uniform.
But suppose we learn

$$h(0) = u, \qquad h(1) = v.$$

The two observations force

$$b = u, \qquad a = v - u.$$

What happens after we observe two hash values?

Let

$$h(x) = ax + b \pmod{p}.$$

Before seeing any outputs, every fixed pair $(h(x), h(y))$ for $x \neq y$ is uniform.
But suppose we learn

$$h(0) = u, \qquad h(1) = v.$$

The two observations force

$$b = u, \qquad a = v - u.$$

So the entire function is now determined. For example,

$$h(2) = 2v - u \pmod{p}.$$

What happens after we observe two hash values?

Let

$$h(x) = ax + b \pmod{p}.$$

Before seeing any outputs, every fixed pair $(h(x), h(y))$ for $x \neq y$ is uniform.
But suppose we learn

$$h(0) = u, \qquad h(1) = v.$$

The two observations force

$$b = u, \qquad a = v - u.$$

So the entire function is now determined. For example,

$$h(2) = 2v - u \pmod{p}.$$

Pairwise independence is an unconditional statement about fixed inputs. It does not imply that unobserved hash values remain uniform or independent after conditioning on observed ones.

Why this matters for two choices

Lecture 6 used the following fully random step. If $H_i(t)$ is the set of bins currently at load at least i, then

$$\Pr(\text{both choices hit } H_i(t) \mid \mathcal{F}_t) = \left(\frac{|H_i(t)|}{n}\right)^2.$$

Why this matters for two choices

Lecture 6 used the following fully random step. If $H_i(t)$ is the set of bins currently at load at least i, then

$$\Pr(\text{both choices hit } H_i(t) \mid \mathcal{F}_t) = \left(\frac{|H_i(t)|}{n}\right)^2.$$

Under fully random choices, this is valid because after conditioning on the current history $\mathcal{F}_t$, the next two bin choices are still independent uniform samples.

Why this matters for two choices

Lecture 6 used the following fully random step. If $H_i(t)$ is the set of bins currently at load at least i, then

$$\Pr(\text{both choices hit } H_i(t) \mid \mathcal{F}_t) = \left(\frac{|H_i(t)|}{n}\right)^2.$$

Under fully random choices, this is valid because after conditioning on the current history $\mathcal{F}_t$, the next two bin choices are still independent uniform samples.

The standard proof assumes that, even after conditioning on the current state, the next two choices are still independent and uniform. Universality does not give us that guarantee.

Can two choices use a practical hash function?

The standard fully random proof does not apply to an arbitrary universal hash family.

Can two choices use a practical hash function?

The standard fully random proof does not apply to an arbitrary universal hash family.
But that does not mean two-choice load balancing requires fully random hash functions.

Can two choices use a practical hash function?

The standard fully random proof does not apply to an arbitrary universal hash family.
But that does not mean two-choice load balancing requires fully random hash functions.
We now look at **simple tabulation hashing**: a very fast family based on table lookups and XOR. Its hash values have real dependencies, but two-choice load balancing still works.

Simple tabulation hashing

Split a machine word into c blocks, traditionally called *characters*:

$$x = (x_1, \ldots, x_c), \qquad x_i \in \{0,1\}^b.$$

For example, a 64-bit key might be split into four 16-bit characters.

Simple tabulation hashing

Split a machine word into c blocks, traditionally called *characters*:

$$x = (x_1, \ldots, x_c), \qquad x_i \in \{0,1\}^b.$$

For example, a 64-bit key might be split into four 16-bit characters.
For each character position i, choose a random table

$$T_i : \{0,1\}^b \to \{0,1\}^\ell.$$

This means that each entry $T_i[z]$ is an independent uniform ℓ-bit string, independently across all i's and all character values z.

Simple tabulation hashing

Split a machine word into c blocks, traditionally called *characters*:

$$x = (x_1, \ldots, x_c), \qquad x_i \in \{0,1\}^b.$$

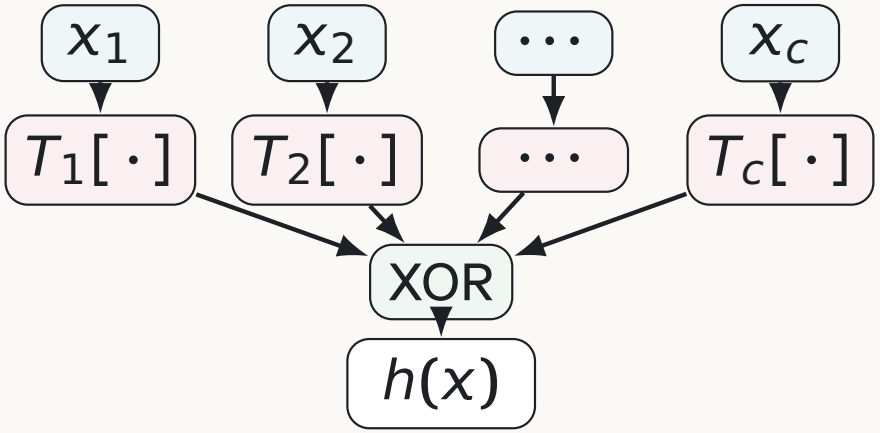
For example, a 64-bit key might be split into four 16-bit characters.
For each character position i, choose a random table

$$T_i : \{0,1\}^b \to \{0,1\}^\ell.$$

This means that each entry $T_i[z]$ is an independent uniform ℓ-bit string, independently across all i's and all character values z.
Define

$$h(x) = T_1[x_1] \oplus \cdots \oplus T_c[x_c].$$

Simple tabulation: evaluation



Evaluation uses c table lookups and $c - 1$ XORs after the tables have been sampled once.

This is usually not the smallest possible seed. Its appeal is fast evaluation and useful probabilistic behavior despite dependencies.

Two choices with simple tabulation

Sample two independent simple-tabulation hash functions

$$h_1, h_2 : U \to [n].$$

For each arriving key x_t:

1. compute the candidate bins

$$b_1 = h_1(x_t), \qquad b_2 = h_2(x_t);$$

2. inspect the current loads of b_1 and b_2;
3. place x_t into the less loaded bin, breaking ties arbitrarily.

Two choices with simple tabulation

Sample two independent simple-tabulation hash functions

$$h_1, h_2 : U \to [n].$$

For each arriving key x_t:

1. compute the candidate bins

$$b_1 = h_1(x_t), \qquad b_2 = h_2(x_t);$$

2. inspect the current loads of b_1 and b_2;
3. place x_t into the less loaded bin, breaking ties arbitrarily.

The hash functions are fixed after initialization. The adaptivity is in the load-balancing process: the placement decision uses the state created by earlier placements.

Simple tabulation has dependent hash values

For two characters,

$$h(a, b) = T_1[a] \oplus T_2[b].$$

Consider four keys:

$$(a, b), \quad (a, b'), \quad (a', b), \quad (a', b').$$

Simple tabulation has dependent hash values

For two characters,

$$h(a, b) = T_1[a] \oplus T_2[b].$$

Consider four keys:

$$(a, b), \quad (a, b'), \quad (a', b), \quad (a', b').$$

Their hash values are

$$\begin{aligned} h(a, b) &= T_1[a] \oplus T_2[b], \\ h(a, b') &= T_1[a] \oplus T_2[b'], \\ h(a', b) &= T_1[a'] \oplus T_2[b], \\ h(a', b') &= T_1[a'] \oplus T_2[b']. \end{aligned}$$

Simple tabulation has dependent hash values

For two characters,

$$h(a, b) = T_1[a] \oplus T_2[b].$$

Consider four keys:

$$(a, b), \quad (a, b'), \quad (a', b), \quad (a', b').$$

Their hash values are

$$\begin{aligned} h(a, b) &= T_1[a] \oplus T_2[b], \\ h(a, b') &= T_1[a] \oplus T_2[b'], \\ h(a', b) &= T_1[a'] \oplus T_2[b], \\ h(a', b') &= T_1[a'] \oplus T_2[b']. \end{aligned}$$

XOR the four equations. Every table entry appears twice, and $z \oplus z = 0$, so

$$h(a, b) \oplus h(a, b') \oplus h(a', b) \oplus h(a', b') = 0.$$

Thus simple tabulation is not 4-wise independent.

The two-choice bound still holds

Dahlgaard, Knudsen, Rotenberg, and Thorup proved that two-choice load balancing with two independent simple-tabulation hash functions still has maximum load

$$O(\log \log n)$$

with high probability for $O(n)$ balls and n bins.

The two-choice bound still holds

Dahlgaard, Knudsen, Rotenberg, and Thorup proved that two-choice load balancing with two independent simple-tabulation hash functions still has maximum load

$$O(\log \log n)$$

with high probability for $O(n)$ balls and n bins.
So the standard fully random proof is not valid for this hash family, but the asymptotic two-choice guarantee still holds.

Recap

$$\Pr(h(x) = h(y)) \leq \frac{1}{m}$$

for each fixed distinct pair x, y is enough for the chained-hashing expectation bound.

- **Affine hashing:** full randomness is unnecessary.
- **Toeplitz hashing:** $w + \ell - 1$ random bits suffice for a universal linear family.
- **Two choices:** simple tabulation still gives $O(\log \log n)$ maximum load.

A convenient proof can require more randomness than the algorithm actually needs.

Next: perfect hashing—worst-case constant lookup for a fixed stored set.

Questions?