Act2Qrypt Virtual Seminar

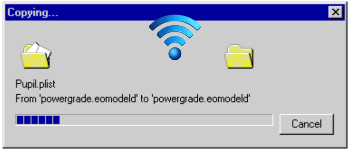
Differentially Private Compression and the Sensitivity of LZ77

February 26, 2026

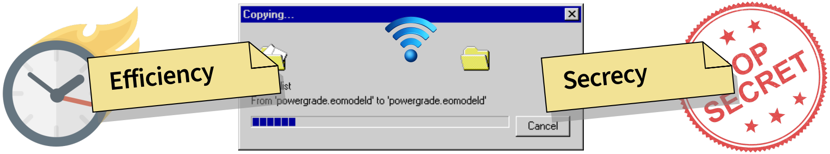
Jeremiah Blocki[1] **Seunghoon Lee**[2] Brayan Sebastián Yepes-Garcia[3]

[1]Purdue University [2]**University of Waterloo** [3]Universidad Nacional de Colombia

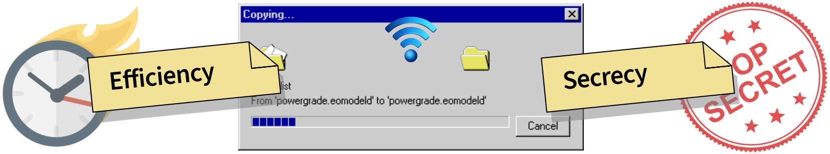
Data Compression and Encryption



Data Compression and Encryption



Data Compression and Encryption

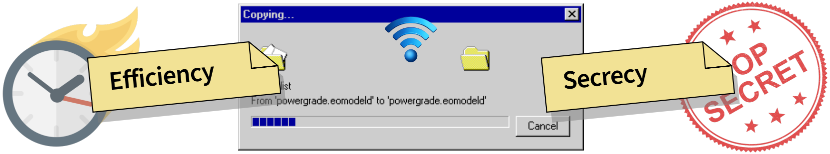


- **Data compression** algorithms exploit redundancy in data

ex) **LZ77 Compression Scheme:**

- Used in ZIP archives and web content compression by Google

Data Compression and Encryption



- **Data compression** algorithms exploit redundancy in data

ex) **LZ77 Compression Scheme:**

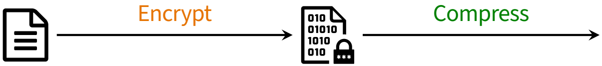
- Used in ZIP archives and web content compression by Google

- **Data encryption** protects the content of plaintext message

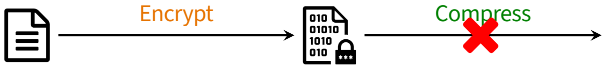
ex) AES-GCM

- Encryption in transit via TLS 1.3

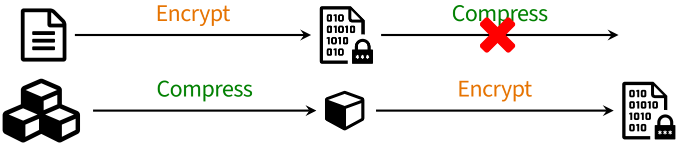
Compress-Then-Encrypt Framework



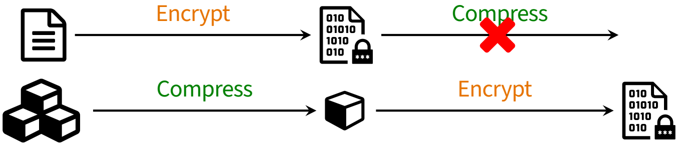
Compress-Then-Encrypt Framework



Compress-Then-Encrypt Framework

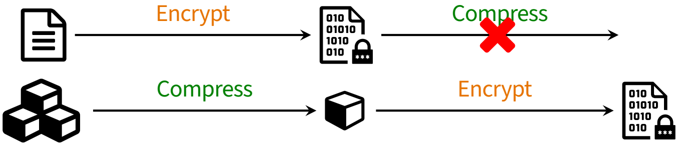


Compress-Then-Encrypt Framework



- Encryption protects the content, but might reveal the length
  - For AES-GCM, the ciphertext leaks the exact length of the underlying plaintext
- This can leak the **content** of the original uncompressed file!

Compress-Then-Encrypt Framework



- Encryption protects the content, but might reveal the length
  - For AES-GCM, the ciphertext leaks the exact length of the underlying plaintext
- This can leak the **content** of the original uncompressed file!

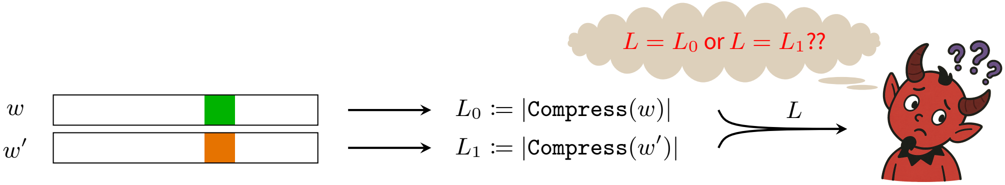
Real-World Attacks.

- **CRIME** (**C**ompression **R**atio **I**nfo-leak **M**ade **E**asy): exploits compression length leakage to hijack TLS sessions
- **BREACH** (**Br**owser **R**econnaissance and **E**xfiltration via **A**daptive **C**ompression of **H**ypertext) attack against the HTTPS protocol exploits compression in the underlying HTTP protocol

*Can one design a **compression scheme** which provides rigorous **privacy guarantees** against an attacker who learns the **length** of the **compressed file?***

Differential Privacy for Compression Schemes (1/2)

We will consider two **neighboring strings** (differing in one symbol; denoted by $w \sim w'$)



Definition

- A compression algorithm $\texttt{Compress}$ is **(ε, δ)-differentially private** if $\forall w \sim w'$ and $\forall S \subseteq \mathbb{N}$,

$$\Pr\left[|\texttt{Compress}(w)| \in S\right] \leq e^{\varepsilon} \cdot \Pr\left[|\texttt{Compress}(w')| \in S\right] + \delta.$$

- Let Σ, Σ' be sets of alphabets. The **global sensitivity** of a compression scheme $\texttt{Compress} : \Sigma^* \to (\Sigma')^*$ for strings of length n is defined as

$$\mathsf{GS}_{\texttt{Compress}}(n) := \max_{w \in \Sigma^n} \max_{w' : w \sim w'} \left| |\texttt{Compress}(w)| - |\texttt{Compress}(w')| \right|.$$

Differential Privacy for Compression Schemes (2/2)

Definition

- A compression algorithm $\texttt{Compress}$ is (ε, δ)-**differentially private** if $\forall w \sim w'$ and $\forall S \subseteq \mathbb{N}$,

$$\Pr\left[|\texttt{Compress}(w)| \in S\right] \leq e^{\varepsilon} \cdot \Pr\left[|\texttt{Compress}(w')| \in S\right] + \delta.$$

- Let Σ, Σ' be sets of alphabets. The **global sensitivity** of a compression scheme $\texttt{Compress} : \Sigma^* \to (\Sigma')^*$ for strings of length n is defined as

$$\mathsf{GS}_{\texttt{Compress}}(n) := \max_{w \in \Sigma^n} \max_{w' : w \sim w'} \Big| |\texttt{Compress}(w)| - |\texttt{Compress}(w')| \Big|.$$

- **Note.** Heal the Breach proposal [PFPSÚGDZ22] does **not** satisfy differential privacy
  - Lagarde and Perifel [LP18]: LZ78 compression is **highly sensitive** to small bit changes
  - There exists $w \in \Sigma^n$ such that $|\texttt{LZ78}(w)| = o(n)$ but $|\texttt{LZ78}(0w)| = \Omega(n)$

Differential Privacy for Compression Schemes (2/2)

Definition

- A compression algorithm $\texttt{Compress}$ is (ε, δ)-**differentially private** if $\forall w \sim w'$ and $\forall S \subseteq \mathbb{N}$,

$$\Pr\left[|\texttt{Compress}(w)| \in S\right] \leq e^{\varepsilon} \cdot \Pr\left[|\texttt{Compress}(w')| \in S\right] + \delta.$$

- Let Σ, Σ' be sets of alphabets. The **global sensitivity** of a compression scheme $\texttt{Compress} : \Sigma^* \to (\Sigma')^*$ for strings of length n is defined as

$$\mathsf{GS}_{\texttt{Compress}}(n) := \max_{w \in \Sigma^n} \max_{w' : w \sim w'} \left| |\texttt{Compress}(w)| - |\texttt{Compress}(w')| \right|.$$

- **Note.** Heal the Breach proposal [PFPSÚGDZ22] does **not** satisfy differential privacy
  - Lagarde and Perifel [LP18]: LZ78 compression is **highly sensitive** to small bit changes
  - There exists $w \in \Sigma^n$ such that $|\texttt{LZ78}(w)| = o(n)$ but $|\texttt{LZ78}(0w)| = \Omega(n)$
  - To prevent distinguishing between w and w', the length of the padding should be **at least** $\Omega(n)$
  - Heal the Breach adds **padding of length** $o(n)$

Does a ***compression scheme*** *with an* ***optimal compression ratio*** *necessarily have* ***high sensitivity?***

Our Contributions

Q1. *Does a compression scheme with an optimal compression ratio necessarily have high sensitivity?*

- **No!** Kolmogorov compression has low global sensitivity: $\mathcal{O}(\log n)$.
- We construct a computable variant of Kolmogorov compression that preserves the global sensitivity, i.e., $\mathcal{O}(\log n)$.

Q2. *Can one design an efficient compression scheme which provides rigorous privacy guarantees against an attacker who learns the length of the compressed file?*

- We give a general framework that transforms any compression scheme into a **differentially private** compression algorithm.
- We give **almost-tight upper/lower bound** for the global sensitivity of the **LZ77 compression algorithm**.
  - Upper bound: $\mathcal{O}(n^{2/3} \log n)$ where n: string length.
    - With bounded sliding window size W, we have an upper bound $\mathcal{O}(W^{2/3} \log n)$.
    - We prove the same GS upper bound for the LZ77 with self-referencing.
  - Lower bound: $\Omega(n^{2/3} \log^{1/3} n)$. (Prior best lower bound: $\Omega(\sqrt{n})$ [AFI23])

Our Contributions

Q1. *Does a compression scheme with an optimal compression ratio necessarily have high sensitivity?*

- **No!** Kolmogorov compression has low global sensitivity: $\mathcal{O}(\log n)$.
- We construct a computable variant of Kolmogorov compression that preserves the global sensitivity, i.e., $\mathcal{O}(\log n)$.

Q2. *Can one design an efficient compression scheme which provides rigorous privacy guarantees against an attacker who learns the length of the compressed file?*

- We give a general framework that transforms any compression scheme into a **differentially private** compression algorithm.
- We give **almost-tight upper/lower bound** for the global sensitivity of the **LZ77 compression algorithm**.
  - Upper bound: $\mathcal{O}(n^{2/3} \log n)$ where n: string length.
    - With bounded sliding window size W, we have an upper bound $\mathcal{O}(W^{2/3} \log n)$.
    - We prove the same GS upper bound for the LZ77 with self-referencing.
  - Lower bound: $\Omega(n^{2/3} \log^{1/3} n)$. (Prior best lower bound: $\Omega(\sqrt{n})$ [AFI23])

Our Contributions

Q1. *Does a compression scheme with an optimal compression ratio necessarily have high sensitivity?*

- **No!** Kolmogorov compression has low global sensitivity: $\mathcal{O}(\log n)$.
- We construct a computable variant of Kolmogorov compression that preserves the global sensitivity, i.e., $\mathcal{O}(\log n)$.

Q2. *Can one design an efficient compression scheme which provides rigorous privacy guarantees against an attacker who learns the length of the compressed file?*

- We give a general framework that transforms any compression scheme into a **differentially private** compression algorithm.

- We give **almost-tight upper/lower bound** for the global sensitivity of the **LZ77 compression algorithm**.
  - Upper bound: $\mathcal{O}(n^{2/3} \log n)$ where n: string length.
    - With bounded sliding window size W, we have an upper bound $\mathcal{O}(W^{2/3} \log n)$.
    - We prove the same GS upper bound for the LZ77 with self-referencing.
  - Lower bound: $\Omega(n^{2/3} \log^{1/3} n)$. (Prior best lower bound: $\Omega(\sqrt{n})$ [AFI23])

How to Construct Differentially Private Compression?

Standard DP technique: add a random amount of padding (p characters)
How much? It depends on the **global sensitivity** of the compression algorithm.

$\texttt{DPCompress}(w, \varepsilon, \delta)$:

$\texttt{Compress}(w)$	$011 \cdots 111$

$$p := \max\{1, \lceil Z + k \rceil\}$$

$$Z \sim \mathrm{Lap}\left(\frac{\mathsf{GS}}{\varepsilon}\right) \qquad k = \mathsf{GS} \cdot \left(\frac{1}{\varepsilon} \ln\left(\frac{1}{2\delta}\right) + 1\right) + 1$$

How to Construct Differentially Private Compression?

Standard DP technique: add a random amount of padding (p characters)
How much? It depends on the **global sensitivity** of the compression algorithm.

$\mathtt{DPCompress}(w, \varepsilon, \delta)$:

$\mathtt{Compress}(w)$	$011 \cdots 111$

$$p := \max\{1, \lceil Z + k \rceil\}$$

$$Z \sim \mathrm{Lap}\left(\frac{\mathsf{GS}}{\varepsilon}\right) \qquad k = \mathsf{GS} \cdot \left(\frac{1}{\varepsilon} \ln\left(\frac{1}{2\delta}\right) + 1\right) + 1$$

Theorem

$\mathtt{DPCompress}(w, \varepsilon, \delta)$ is (ε, δ)-differentially private.

- k is proportional to $\mathsf{GS}_{\mathtt{Compress}}$ (not proportional to n).
- It is important to understand $\mathsf{GS}_{\mathtt{Compress}}$ for widely used compression schemes.

Practicality of DPCompress

Recall. We add a padding of length $p := \max\{1, \lceil Z + k \rceil\}$ where $Z \sim \mathrm{Lap}(\mathsf{GS}/\varepsilon)$.

On average, the amount of padding is approximately $\mathbb{E}[p] \approx k = \mathsf{GS} \cdot \left(\frac{1}{\varepsilon} \ln\left(\frac{1}{2\delta}\right) + 1\right) + 1$.

- If $k = o(n)$, we achieve efficient compression ratio:

$$\frac{|\texttt{DPCompress}(w, \varepsilon, \delta)|}{n} = \frac{|\texttt{Compress}(w)| + k}{n} = \frac{|\texttt{Compress}(w)|}{n} + o(1).$$

- If $k = \Omega(n)$, it becomes impractical to use `DPCompress` transformation.

Practicality of DPCompress

Recall. We add a padding of length $p := \max\{1, \lceil Z + k \rceil\}$ where $Z \sim \mathrm{Lap}(\mathsf{GS}/\varepsilon)$.

On average, the amount of padding is approximately $\mathbb{E}[p] \approx k = \mathsf{GS} \cdot \left(\frac{1}{\varepsilon} \ln\left(\frac{1}{2\delta}\right) + 1\right) + 1$.

- If $k = o(n)$, we achieve efficient compression ratio:

$$\frac{|\mathtt{DPCompress}(w, \varepsilon, \delta)|}{n} = \frac{|\mathtt{Compress}(w)| + k}{n} = \frac{|\mathtt{Compress}(w)|}{n} + o(1).$$

- If $k = \Omega(n)$, it becomes impractical to use $\mathtt{DPCompress}$ transformation.

Question

Can we find practical compression schemes with global sensitivity $\mathsf{GS} = o(n)$?

- **Recall.** LZ78 does not satisfy low sensitivity [LP18].
- This motivates our study of the global sensitivity of the LZ77 compression scheme.
  - **Almost-tight upper and lower bound** for the **global sensitivity of LZ77** that are $o(n)$.

LZ77 Compression Scheme

- **Input:** a string $w \in \Sigma^n$ where Σ: a set of alphabets
- **Output:** a sequence of blocks $(B_1, \ldots, B_t)$
- We maintain a counter ctc: we have encoded ctc characters so far

$$B_i = [q_i, \ell_i, c_i]$$

- q_i: starting position of the **longest substring** of $w[1, \text{ctc}]$ that we will copy over
- ℓ_i: length of that substring
- c_i: the character we add to the block

LZ77 Compression Scheme

- **Input:** a string $w \in \Sigma^n$ where Σ: a set of alphabets
- **Output:** a sequence of blocks $(B_1, \ldots, B_t)$
- We maintain a counter ctc: we have encoded ctc characters so far

$$B_i = [q_i, \ell_i, c_i]$$

- q_i: starting position of the **longest substring** of $w[1, \text{ctc}]$ that we will copy over
- ℓ_i: length of that substring
- c_i: the character we add to the block

Example. Compression for $w = aaaaaaaaaaaaaab$ (14 a's and one b):

Encoded so far

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
w	a	a	a	a	a	a	a	a	a	a	a	a	a	a	b
	▲ ctc														

$B_1 = [0, 0, a]$

LZ77 Compression Scheme

- **Input:** a string $w \in \Sigma^n$ where Σ: a set of alphabets
- **Output:** a sequence of blocks $(B_1, \ldots, B_t)$
- We maintain a counter ctc: we have encoded ctc characters so far

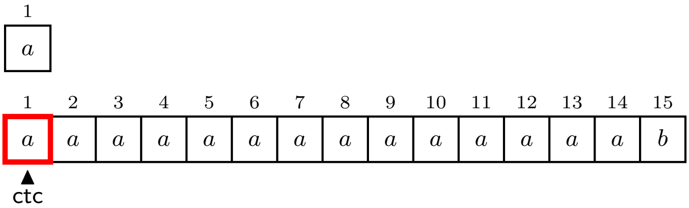
$$B_i = [q_i, \ell_i, c_i]$$

- q_i: starting position of the **longest substring** of $w[1, \text{ctc}]$ that we will copy over
- ℓ_i: length of that substring
- c_i: the character we add to the block

Example. Compression for $w = aaaaaaaaaaaaaab$ (14 a's and one b):

Encoded so far

w



$B_1 = [0, 0, a]$

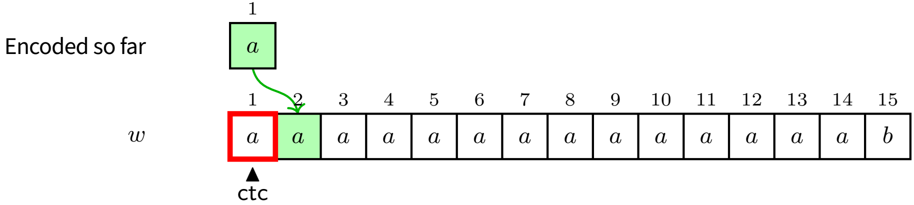
LZ77 Compression Scheme

- **Input:** a string $w \in \Sigma^n$ where Σ: a set of alphabets
- **Output:** a sequence of blocks $(B_1, \ldots, B_t)$
- We maintain a counter ctc: we have encoded ctc characters so far

$$B_i = [q_i, \ell_i, c_i]$$

- q_i: starting position of the **longest substring** of $w[1, \text{ctc}]$ that we will copy over
- ℓ_i: length of that substring
- c_i: the character we add to the block

Example. Compression for $w = aaaaaaaaaaaaaab$ (14 a's and one b):



$B_1 = [0, 0, a]$

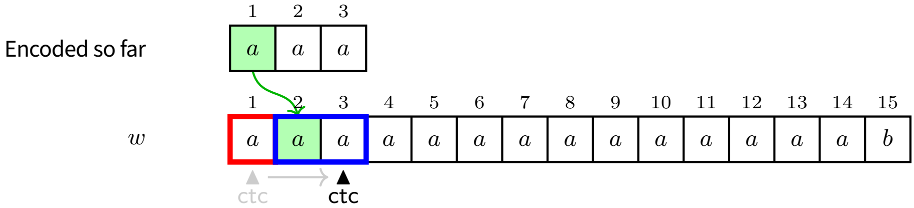
LZ77 Compression Scheme

- **Input:** a string $w \in \Sigma^n$ where Σ: a set of alphabets
- **Output:** a sequence of blocks $(B_1, \ldots, B_t)$
- We maintain a counter ctc: we have encoded ctc characters so far

$$B_i = [q_i, \ell_i, c_i]$$

- q_i: starting position of the **longest substring** of $w[1, \text{ctc}]$ that we will copy over
- ℓ_i: length of that substring
- c_i: the character we add to the block

Example. Compression for $w = aaaaaaaaaaaaaab$ (14 a's and one b):



$B_1 = [0, 0, a]$ $B_2 = [1, 1, a]$

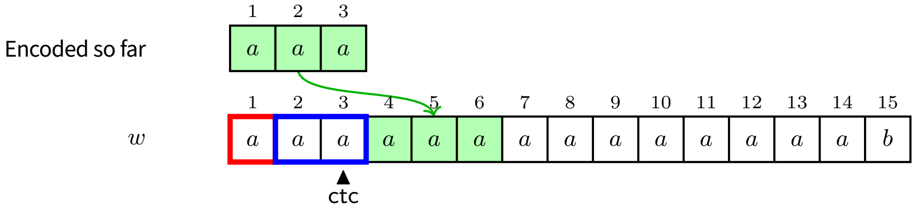
LZ77 Compression Scheme

- **Input:** a string $w \in \Sigma^n$ where Σ: a set of alphabets
- **Output:** a sequence of blocks $(B_1, \ldots, B_t)$
- We maintain a counter ctc: we have encoded ctc characters so far

$$B_i = [q_i, \ell_i, c_i]$$

- q_i: starting position of the **longest substring** of $w[1, \text{ctc}]$ that we will copy over
- ℓ_i: length of that substring
- c_i: the character we add to the block

Example. Compression for $w = aaaaaaaaaaaaaab$ (14 a's and one b):



$B_1 = [0, 0, a]$ $B_2 = [1, 1, a]$

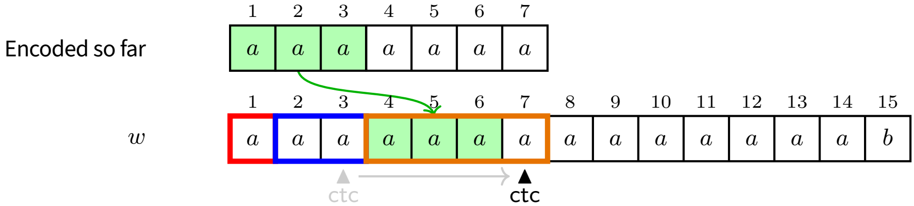
LZ77 Compression Scheme

- **Input:** a string $w \in \Sigma^n$ where Σ: a set of alphabets
- **Output:** a sequence of blocks $(B_1, \ldots, B_t)$
- We maintain a counter ctc: we have encoded ctc characters so far

$$B_i = [q_i, \ell_i, c_i]$$

- q_i: starting position of the **longest substring** of $w[1, \mathsf{ctc}]$ that we will copy over
- ℓ_i: length of that substring
- c_i: the character we add to the block

Example. Compression for $w = aaaaaaaaaaaaaab$ (14 a's and one b):



$B_1 = [0, 0, a]$ $B_2 = [1, 1, a]$ $B_3 = [1, 3, a]$

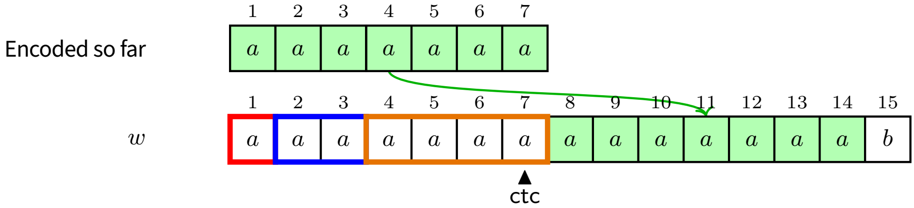
LZ77 Compression Scheme

- **Input:** a string $w \in \Sigma^n$ where Σ: a set of alphabets
- **Output:** a sequence of blocks $(B_1, \ldots, B_t)$
- We maintain a counter ctc: we have encoded ctc characters so far

$$B_i = [q_i, \ell_i, c_i]$$

- q_i: starting position of the **longest substring** of $w[1, \text{ctc}]$ that we will copy over
- ℓ_i: length of that substring
- c_i: the character we add to the block

Example. Compression for $w = aaaaaaaaaaaaaab$ (14 a's and one b):



$B_1 = [0, 0, a]$ $B_2 = [1, 1, a]$ $B_3 = [1, 3, a]$

LZ77 Compression Scheme

- **Input:** a string $w \in \Sigma^n$ where Σ: a set of alphabets
- **Output:** a sequence of blocks $(B_1, \ldots, B_t)$
- We maintain a counter ctc: we have encoded ctc characters so far

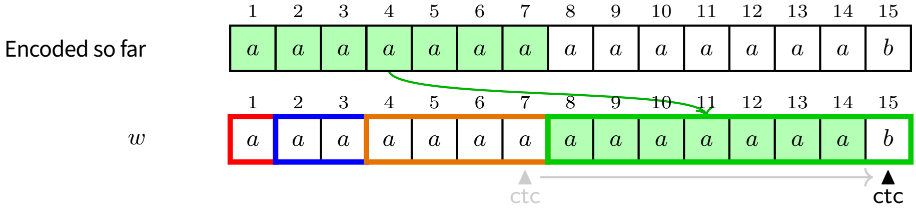
$$B_i = [q_i, \ell_i, c_i]$$

q_i: starting position of the **longest substring** of $w[1, \text{ctc}]$ that we will copy over

ℓ_i: length of that substring

c_i: the character we add to the block

Example. Compression for $w = aaaaaaaaaaaaaab$ (14 a's and one b):



$B_1 = [0, 0, a]$ $B_2 = [1, 1, a]$ $B_3 = [1, 3, a]$ $B_4 = [1, 7, b]$

Upper Bound for the Global Sensitivity of LZ77

- Let $\texttt{LZ77}(w) = \underbrace{(B_1, \ldots, B_t)}_{t \text{ blocks}}$ and $\texttt{LZ77}(w') = \underbrace{(B'_1, \ldots, B'_{t'})}_{t' \text{ blocks}}$
- $B_i = [q_i, \ell_i, c_i]$ such that $0 \leq q_i, \ell_i < n$ and $c_i \in \Sigma$
  - It takes $2\lceil \log n \rceil + \lceil \log |\Sigma| \rceil$ bits to encode each block
- It is crucial to understand the **upper bound** of $|t' - t|$ (i.e., the difference of the number of blocks)

$$
\begin{aligned}
(\because)\ \mathsf{GS}_{\mathrm{LZ77}}(n) &:= \max_{w \in \Sigma^n} \max_{w' : w \sim w'} \Big| |\texttt{LZ77}(w)| - |\texttt{LZ77}(w')| \Big| \\
&= \max_{w \in \Sigma^n} \max_{w' : w \sim w'} \big| t' - t \big| \left(2\lceil \log n \rceil + \lceil \log |\Sigma| \rceil\right) \\
&= \left(2\lceil \log n \rceil + \lceil \log |\Sigma| \rceil\right) \cdot \max_{w \in \Sigma^n} \max_{w' : w \sim w'} \big| t' - t \big|
\end{aligned}
$$

Upper Bound for the Global Sensitivity of LZ77

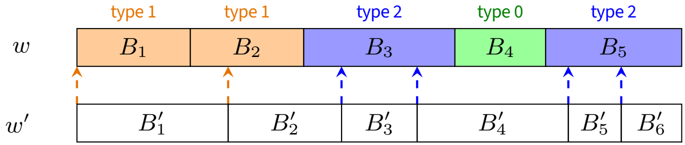
- Let $\texttt{LZ77}(w) = \underbrace{(B_1, \ldots, B_t)}_{t \text{ blocks}}$ and $\texttt{LZ77}(w') = \underbrace{(B'_1, \ldots, B'_{t'})}_{t' \text{ blocks}}$
- $B_i = [q_i, \ell_i, c_i]$ such that $0 \leq q_i, \ell_i < n$ and $c_i \in \Sigma$
  - It takes $2\lceil \log n \rceil + \lceil \log |\Sigma| \rceil$ bits to encode each block
- It is crucial to understand the **upper bound** of $|t' - t|$ (i.e., the difference of the number of blocks)

$$
\begin{aligned}
(\because)\ \mathsf{GS}_{\texttt{LZ77}}(n) &:= \max_{w \in \Sigma^n} \max_{w' : w \sim w'} \big| |\texttt{LZ77}(w)| - |\texttt{LZ77}(w')| \big| \\
&= \max_{w \in \Sigma^n} \max_{w' : w \sim w'} |t' - t| \left(2\lceil \log n \rceil + \lceil \log |\Sigma| \rceil\right) \\
&= \left(2\lceil \log n \rceil + \lceil \log |\Sigma| \rceil\right) \cdot \max_{w \in \Sigma^n} \max_{w' : w \sim w'} |t' - t|
\end{aligned}
$$

Note. We will abuse the notation that B_i can also denote the actual string that represents the block.

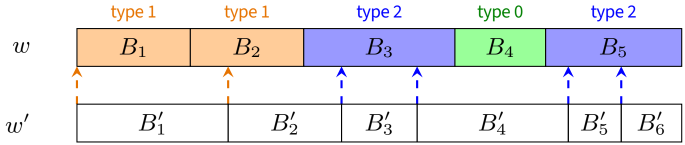
Combinatorial Property of the Blocks

- $\texttt{LZ77}(w) = (B_1, \ldots, B_t), \texttt{LZ77}(w') = (B'_1, \ldots, B'_{t'})$
- We can split the set of blocks $\{B_1, \ldots, B_t\}$ in $\texttt{LZ77}(w)$ based on the **number of blocks (of w')** that **start inside B_i**.



Combinatorial Property of the Blocks

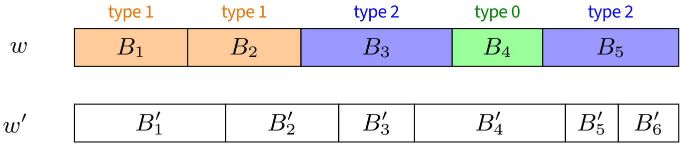
- $\texttt{LZ77}(w) = (B_1, \ldots, B_t), \texttt{LZ77}(w') = (B'_1, \ldots, B'_{t'})$
- We can split the set of blocks $\{B_1, \ldots, B_t\}$ in $\texttt{LZ77}(w)$ based on the **number of blocks (of w')** that **start inside B_i**.



- There exist only **type-0, type-1,** and **type-2** blocks for $\texttt{LZ77}(w)$. [AFI23]

Combinatorial Property of the Blocks

- $\texttt{LZ77}(w) = (B_1, \ldots, B_t), \texttt{LZ77}(w') = (B'_1, \ldots, B'_{t'})$
- We can split the set of blocks $\{B_1, \ldots, B_t\}$ in $\texttt{LZ77}(w)$ based on the **number of blocks (of w')** that **start inside B_i**.

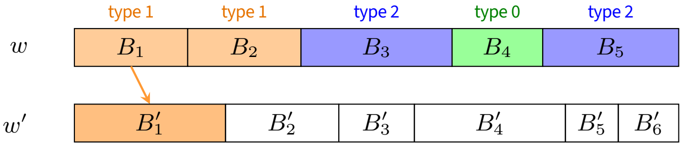


- There exist only **type-0, type-1,** and **type-2** blocks for $\texttt{LZ77}(w)$. [AFI23]
- Let t_m denote the number of type-m blocks. Then we observe:

$$t = t_0 + t_1 + t_2, \text{ and}$$

Combinatorial Property of the Blocks

- $\texttt{LZ77}(w) = (B_1, \ldots, B_t), \texttt{LZ77}(w') = (B'_1, \ldots, B'_{t'})$
- We can split the set of blocks $\{B_1, \ldots, B_t\}$ in $\texttt{LZ77}(w)$ based on the **number of blocks (of w')** that **start inside B_i**.

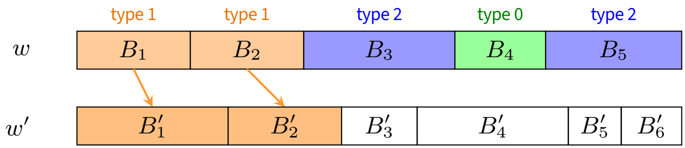


- There exist only **type-0, type-1,** and **type-2** blocks for $\texttt{LZ77}(w)$. [AFI23]
- Let t_m denote the number of type-m blocks. Then we observe:

$$t = t_0 + t_1 + t_2, \text{ and}$$
$$t' = 0 \cdot t_0 + 1 \cdot t_1 + 2 \cdot t_2.$$

Combinatorial Property of the Blocks

- $\texttt{LZ77}(w) = (B_1, \ldots, B_t), \texttt{LZ77}(w') = (B'_1, \ldots, B'_{t'})$
- We can split the set of blocks $\{B_1, \ldots, B_t\}$ in $\texttt{LZ77}(w)$ based on the **number of blocks (of w')** that **start inside B_i**.

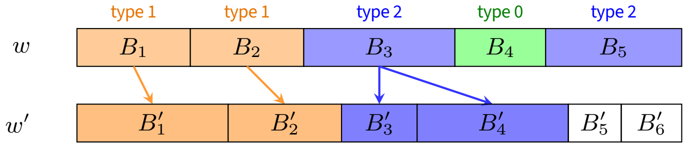


- There exist only **type-0, type-1,** and **type-2** blocks for $\texttt{LZ77}(w)$. [AFI23]
- Let t_m denote the number of type-m blocks. Then we observe:

$$t = t_0 + t_1 + t_2, \text{ and}$$
$$t' = 0 \cdot t_0 + 1 \cdot t_1 + 2 \cdot t_2.$$

Combinatorial Property of the Blocks

- $\texttt{LZ77}(w) = (B_1, \ldots, B_t), \texttt{LZ77}(w') = (B'_1, \ldots, B'_{t'})$
- We can split the set of blocks $\{B_1, \ldots, B_t\}$ in $\texttt{LZ77}(w)$ based on the **number of blocks (of w')** that **start inside B_i**.

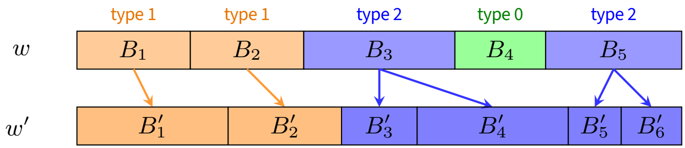


- There exist only **type-0, type-1,** and **type-2** blocks for $\texttt{LZ77}(w)$. [AFI23]
- Let t_m denote the number of type-m blocks. Then we observe:

$$t = t_0 + t_1 + t_2, \text{ and}$$
$$t' = 0 \cdot t_0 + 1 \cdot t_1 + 2 \cdot t_2.$$

Combinatorial Property of the Blocks

- $\texttt{LZ77}(w) = (B_1, \ldots, B_t), \texttt{LZ77}(w') = (B'_1, \ldots, B'_{t'})$
- We can split the set of blocks $\{B_1, \ldots, B_t\}$ in $\texttt{LZ77}(w)$ based on the **number of blocks (of w')** that **start inside B_i**.

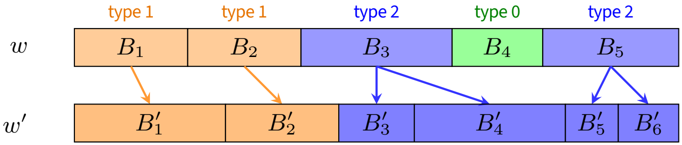


- There exist only **type-0, type-1,** and **type-2** blocks for $\texttt{LZ77}(w)$. [AFI23]
- Let t_m denote the number of type-m blocks. Then we observe:

$$t = t_0 + t_1 + t_2, \text{ and}$$
$$t' = 0 \cdot t_0 + 1 \cdot t_1 + 2 \cdot t_2.$$

Combinatorial Property of the Blocks

- $\texttt{LZ77}(w) = (B_1, \ldots, B_t), \texttt{LZ77}(w') = (B'_1, \ldots, B'_{t'})$
- We can split the set of blocks $\{B_1, \ldots, B_t\}$ in $\texttt{LZ77}(w)$ based on the **number of blocks (of w')** that **start inside** B_i.



- There exist only **type-0, type-1,** and **type-2** blocks for $\texttt{LZ77}(w)$. [AFI23]
- Let t_m denote the number of type-m blocks. Then we observe:

$$t = t_0 + t_1 + t_2, \text{ and}$$
$$t' = 0 \cdot t_0 + 1 \cdot t_1 + 2 \cdot t_2.$$

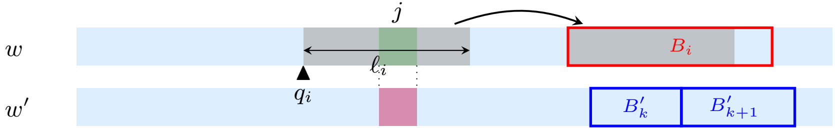
- Hence,

$$t' - t = t_2 - t_0 \leq t_2.$$

$\therefore$ We need to upper bound t_2 (the number of type-2 blocks).

Number of Type-2 Blocks (1/2)

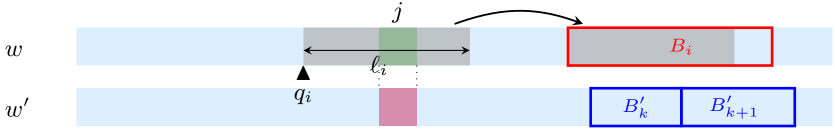
Observe that, the unique differing index j must be contained in the copied substring of any type-2 block.
(except for one edge case)



Number of Type-2 Blocks (1/2)

Observe that, the unique differing index j must be contained in the copied substring of any type-2 block.

(except for one edge case)

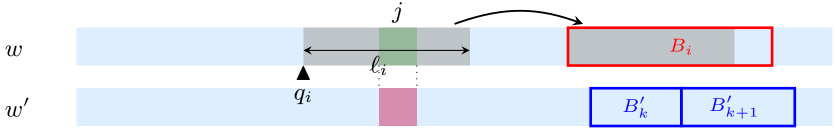


Key Observation

All the type-2 blocks have the **unique (q_i, ℓ_i) pair**. Why?

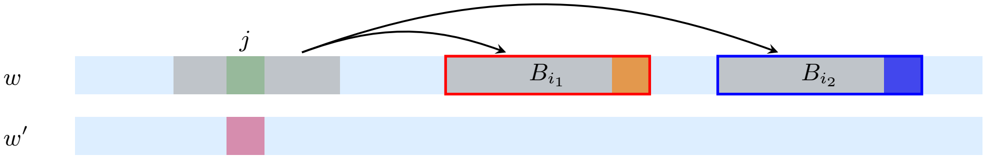
Number of Type-2 Blocks (1/2)

Observe that, the unique differing index j must be contained in the copied substring of any type-2 block.
(except for one edge case)



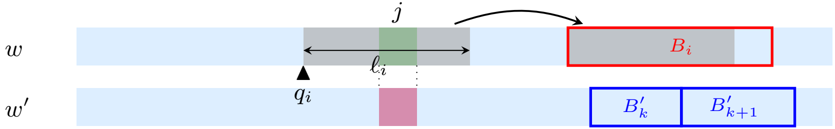
Key Observation

All the type-2 blocks have the **unique (q_i, ℓ_i) pair**. Why?



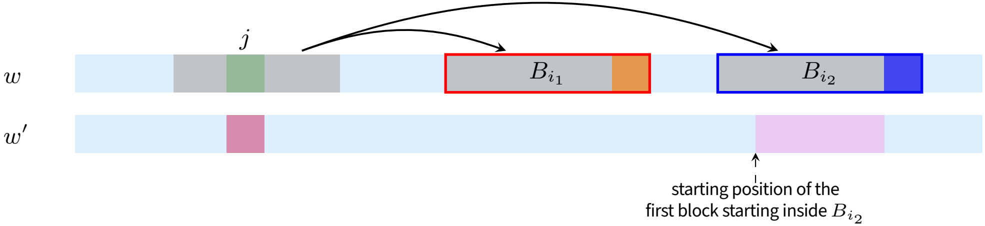
Number of Type-2 Blocks (1/2)

Observe that, the unique differing index j must be contained in the copied substring of any type-2 block.
(except for one edge case)



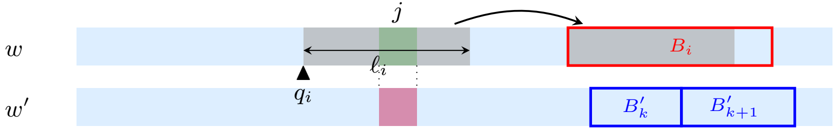
Key Observation

All the type-2 blocks have the **unique (q_i, ℓ_i) pair**. Why?



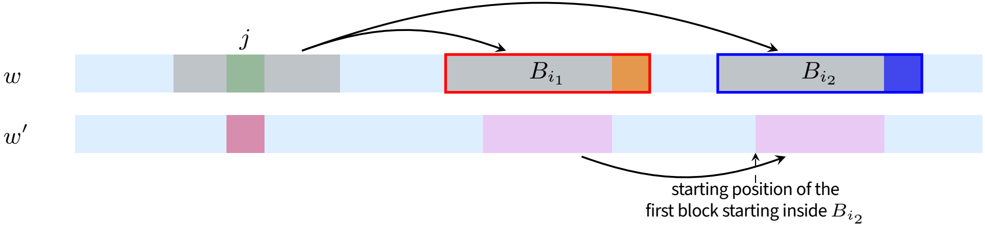
Number of Type-2 Blocks (1/2)

Observe that, the unique differing index j must be contained in the copied substring of any type-2 block.
(except for one edge case)



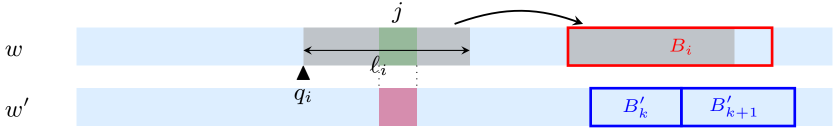
Key Observation

All the type-2 blocks have the **unique (q_i, ℓ_i) pair**. Why?



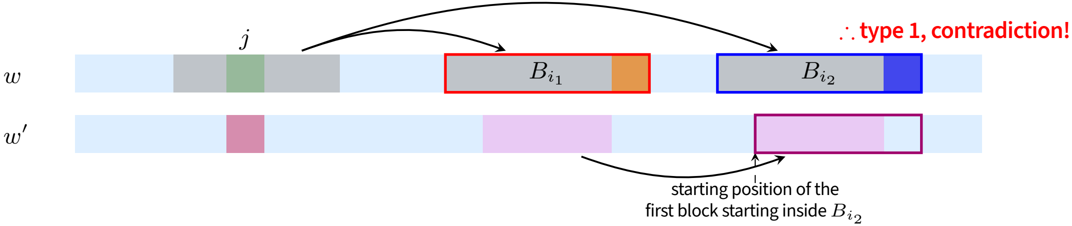
Number of Type-2 Blocks (1/2)

Observe that, the unique differing index j must be contained in the copied substring of any type-2 block.
(except for one edge case)



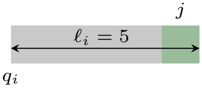
Key Observation

All the type-2 blocks have the **unique (q_i, ℓ_i) pair**. Why?



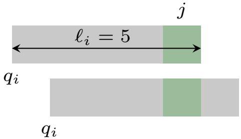
Number of Type-2 Blocks (2/2)

How many type-2 blocks **with a fixed length ℓ?**
Let $x_\ell := \#(\text{type-2 blocks with } \ell_i = \ell)$.



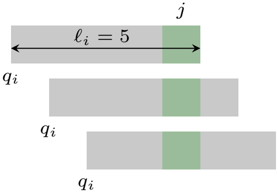
Number of Type-2 Blocks (2/2)

How many type-2 blocks **with a fixed length ℓ?**
Let $x_\ell :=$ #(type-2 blocks with $\ell_i = \ell$).



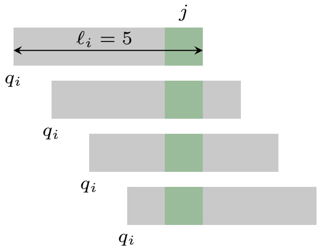
Number of Type-2 Blocks (2/2)

How many type-2 blocks **with a fixed length ℓ?**
Let $x_\ell := \#(\text{type-2 blocks with } \ell_i = \ell)$.



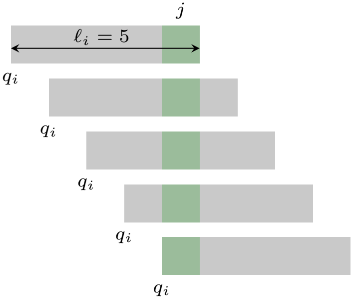
Number of Type-2 Blocks (2/2)

How many type-2 blocks **with a fixed length ℓ?**
Let $x_\ell := \#(\text{type-2 blocks with } \ell_i = \ell)$.



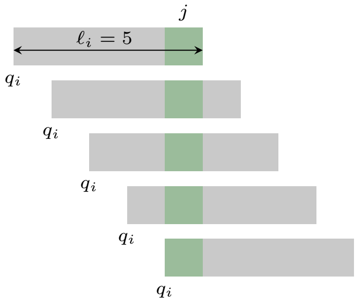
Number of Type-2 Blocks (2/2)

How many type-2 blocks **with a fixed length ℓ?**
Let $x_\ell := \#(\text{type-2 blocks with } \ell_i = \ell)$.



Number of Type-2 Blocks (2/2)

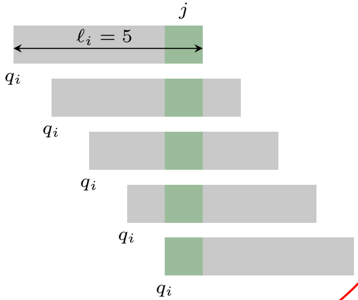
How many type-2 blocks **with a fixed length ℓ?**
Let $x_\ell := \#(\text{type-2 blocks with } \ell_i = \ell)$.



- Uniqueness of $(q_i, \ell) \;\Rightarrow\; x_\ell \le \ell$.
- (Total length of type-2 blocks) $\le n \;\Rightarrow\; \sum_\ell x_\ell(\ell + 1) \le n$.

Number of Type-2 Blocks (2/2)

How many type-2 blocks **with a fixed length ℓ?**
Let $x_\ell := \#(\text{type-2 blocks with } \ell_i = \ell)$.



- Uniqueness of $(q_i, \ell) \Rightarrow x_\ell \leq \ell$.
- (Total length of type-2 blocks) $\leq n \Rightarrow \sum_\ell x_\ell(\ell+1) \leq n$.

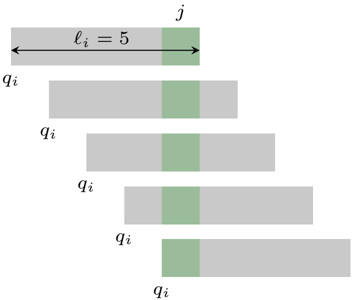
Optimization Problem

Maximize $t_2 = \sum_\ell x_\ell$ subject to

- $x_\ell \leq \ell$, and
- $\sum_\ell x_\ell(\ell+1) \leq n$.

Number of Type-2 Blocks (2/2)

How many type-2 blocks **with a fixed length ℓ?**
Let $x_\ell := \#(\text{type-2 blocks with } \ell_i = \ell)$.



- Uniqueness of $(q_i, \ell) \;\Rightarrow\; x_\ell \leq \ell$.
- (Total length of type-2 blocks) $\leq n \;\Rightarrow\; \sum_\ell x_\ell(\ell+1) \leq n$.

Optimization Problem

Maximize $t_2 = \sum_\ell x_\ell$ subject to

- $x_\ell \leq \ell$, and
- $\sum_\ell x_\ell(\ell+1) \leq n$.

Solving the optimization problem gives

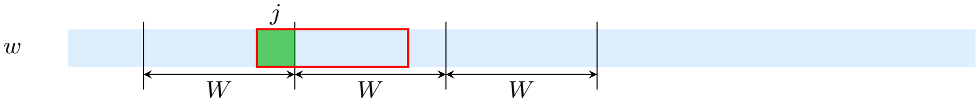
$$t_2 = \mathcal{O}(n^{2/3}),$$

and therefore

$$\begin{aligned}\mathrm{GS}_{\mathrm{LZ77}}(n) &\leq t_2 \cdot \mathcal{O}(\log n)\\ &= \mathcal{O}(n^{2/3}\log n).\end{aligned}$$

GS Upper Bound for LZ77: with Sliding Window $W < n$

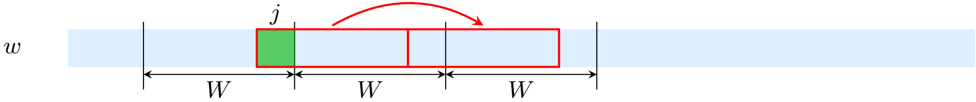
Without sliding window $W < n$, total length of type-2 blocks was $\leq n$, i.e., $\sum_\ell x_\ell(\ell+1) \leq n$.



- There is no type-2 block starting after $j + W$

GS Upper Bound for LZ77: with Sliding Window $W < n$

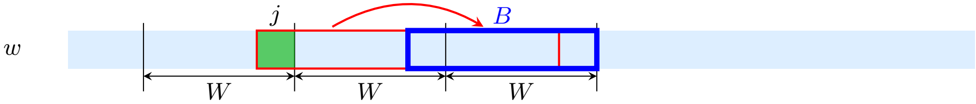
Without sliding window $W < n$, total length of type-2 blocks was $\leq n$, i.e., $\sum_\ell x_\ell(\ell + 1) \leq n$.



- There is no type-2 block starting after $j + W$

GS Upper Bound for LZ77: with Sliding Window $W < n$

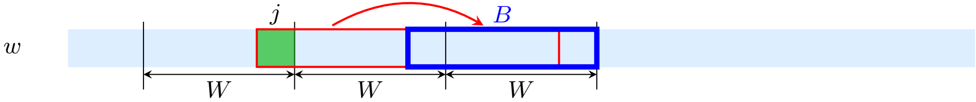
Without sliding window $W < n$, total length of type-2 blocks was $\leq n$, i.e., $\sum_\ell x_\ell(\ell+1) \leq n$.



- There is no type-2 block starting after $j + W$

GS Upper Bound for LZ77: with Sliding Window $W < n$

Without sliding window $W < n$, total length of type-2 blocks was $\leq n$, i.e., $\sum_\ell x_\ell(\ell+1) \leq n$.



- There is no type-2 block starting after $j + W$

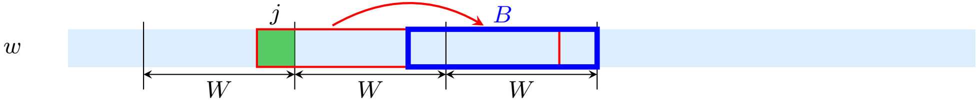
Optimization Problem

Maximize $t_2 = \sum_\ell x_\ell$ subject to

- $x_\ell \leq \ell$, and
- $\sum_\ell x_\ell(\ell+1) \leq 3W$.

GS Upper Bound for LZ77: with Sliding Window $W < n$

Without sliding window $W < n$, total length of type-2 blocks was $\leq n$, i.e., $\sum_\ell x_\ell(\ell+1) \leq n$.



- There is no type-2 block starting after $j + W$

Optimization Problem

Maximize $t_2 = \sum_\ell x_\ell$ subject to

- $x_\ell \leq \ell$, and
- $\sum_\ell x_\ell(\ell+1) \leq 3W$.

Solving the optimization problem gives

$$t_2 = \mathcal{O}(W^{2/3}),$$

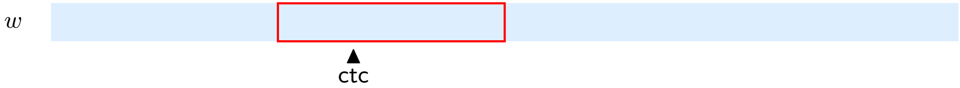
and therefore

$$\begin{aligned} \text{GS}_{\text{LZ77}}(n) &\leq t_2 \cdot \mathcal{O}(\log n) \\ &= \mathcal{O}(W^{2/3} \log n). \end{aligned}$$

LZ77 with Self-Referencing

LZ77 with self-referencing allows overlapping matches.

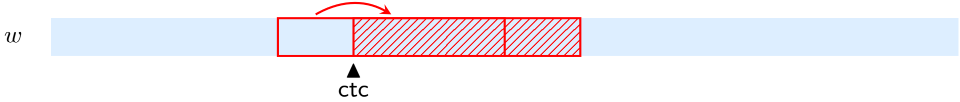
- DEFLATE (zip/png): RFC 1951 [Deu96] allows self-referencing



LZ77 with Self-Referencing

LZ77 with self-referencing allows overlapping matches.

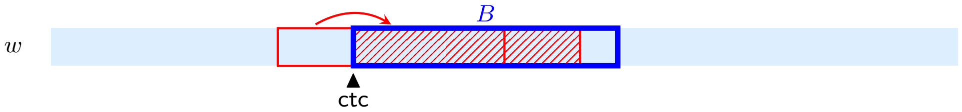
- DEFLATE (zip/png): RFC 1951 [Deu96] allows self-referencing



LZ77 with Self-Referencing

LZ77 with self-referencing allows overlapping matches.

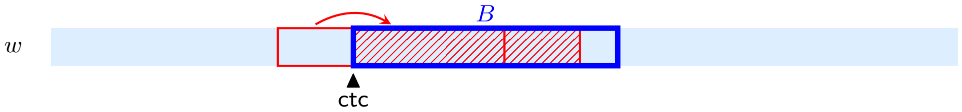
- DEFLATE (zip/png): RFC 1951 [Deu96] allows self-referencing



LZ77 with Self-Referencing

LZ77 with self-referencing allows overlapping matches.

- DEFLATE (zip/png): RFC 1951 [Deu96] allows self-referencing



Example. Compression for $w = aaaaaaaaaaaaaab$ (14 a's and one b):

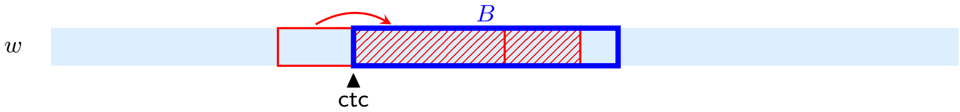
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
w	a	a	a	a	a	a	a	a	a	a	a	a	a	a	b

$$B_1 = [0, 0, a]$$

LZ77 with Self-Referencing

LZ77 with self-referencing allows overlapping matches.

- DEFLATE (zip/png): RFC 1951 [Deu96] allows self-referencing



Example. Compression for $w = aaaaaaaaaaaaaab$ (14 a's and one b):

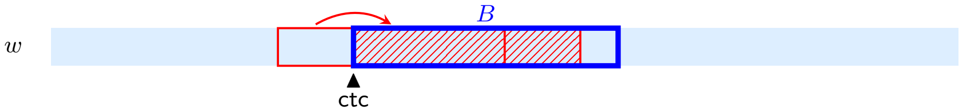
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
w	a	a	a	a	a	a	a	a	a	a	a	a	a	a	b

$B_1 = [0, 0, a]$

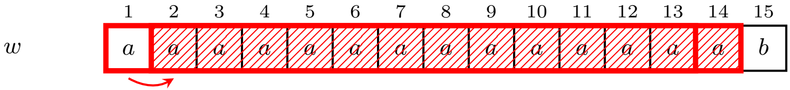
LZ77 with Self-Referencing

LZ77 with self-referencing allows overlapping matches.

- DEFLATE (zip/png): RFC 1951 [Deu96] allows self-referencing



Example. Compression for $w = aaaaaaaaaaaaaab$ (14 a's and one b):

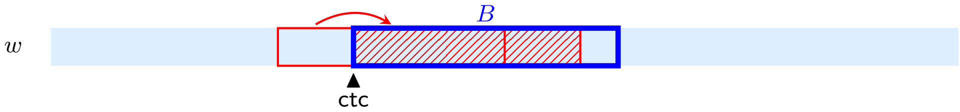


$B_1 = [0, 0, a]$

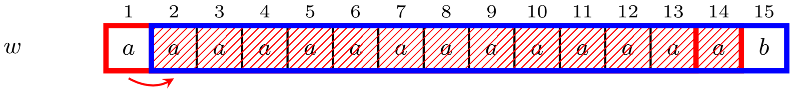
LZ77 with Self-Referencing

LZ77 with self-referencing allows overlapping matches.

- DEFLATE (zip/png): RFC 1951 [Deu96] allows self-referencing



Example. Compression for $w = aaaaaaaaaaaaaab$ (14 a's and one b):

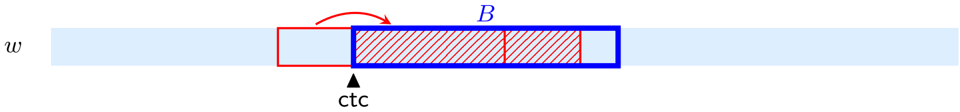


$B_1 = [0, 0, a]$ $B_2 = [1, 13, b]$

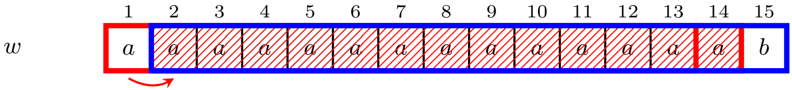
LZ77 with Self-Referencing

LZ77 with self-referencing allows overlapping matches.

- DEFLATE (zip/png): RFC 1951 [Deu96] allows self-referencing



Example. Compression for $w = aaaaaaaaaaaaaab$ (14 a's and one b):



$B_1 = [0, 0, a]$ $B_2 = [1, 13, b]$

Theorem (informal)

For LZ77 with self-referencing with the sliding window size W, the global sensitivity is also $\mathcal{O}(W^{2/3} \log n)$.

GS Lower Bound for LZ77: String Construction

Recall

For LZ77$(w_1) = (B_1, \ldots, B_t)$ and LZ77$(w_2) = (B'_1, \ldots, B'_{t'})$, we have $t' - t = t_2 - t_0,$ where t_i denotes the number of type-i blocks.

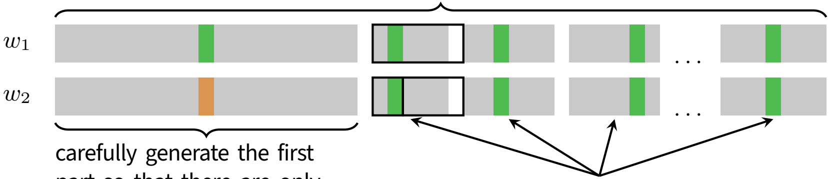
GS Lower Bound for LZ77: String Construction

Recall

For LZ77$(w_1) = (B_1, \ldots, B_t)$ and LZ77$(w_2) = (B'_1, \ldots, B'_{t'})$, we have $t' - t = t_2 - t_0$, where t_i denotes the number of type-i blocks.

High-Level Idea.

(string length) $\approx m^3 \log m$



carefully generate the first part so that there are only type-1 blocks

unique copies from the first part
mostly type 2! & **no type 0**

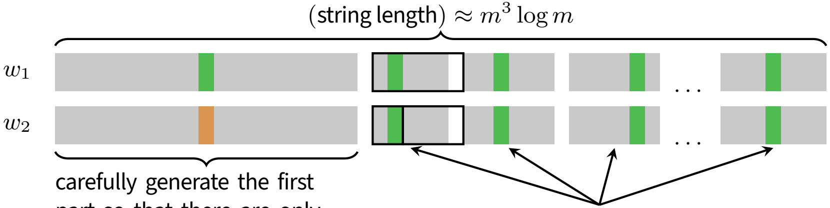
$t_2 = \Theta(m^2/2)$ & $t_0 = 0$

GS Lower Bound for LZ77: String Construction

Recall

For $\texttt{LZ77}(w_1) = (B_1, \ldots, B_t)$ and $\texttt{LZ77}(w_2) = (B'_1, \ldots, B'_{t'})$, we have $t' - t = t_2 - t_0$, where t_i denotes the number of type-i blocks.

High-Level Idea.



carefully generate the first part so that there are only type-1 blocks

unique copies from the first part
mostly type 2! & **no type 0**

$\therefore \left| |\texttt{LZ77}(w_1)| - |\texttt{LZ77}(w_2)| \right|$

$= \Omega(n^{2/3} \log^{1/3} n)$ for $n = m^3 \log m$

$t_2 = \Theta(m^2/2)$ & $t_0 = 0$

Conclusion

- General framework to convert any compression scheme to **differentially private** compression scheme
  - Random amount of padding proportional to the **global sensitivity** of the compression scheme
- **Almost-tight upper and lower bound** for the global sensitivity of LZ77 compression
  - $\mathrm{GS}_{\mathrm{LZ77}}(n) = \mathcal{O}(n^{2/3} \log n)$ and $\mathrm{GS}_{\mathrm{LZ77}}(n) = \Omega(n^{2/3} \log^{1/3} n)$
  - With sliding window size W, $\mathrm{GS}_{\mathrm{LZ77}}(n) = \mathcal{O}(W^{2/3} \log n)$
  - Same upper bound holds for LZ77 with self-referencing (which allows overlapping matches)
- Since $\mathrm{GS}_{\mathrm{LZ77}}(n) = o(n)$, we can argue that **LZ77 is practical for DPCompress**:

$$\frac{|\mathtt{DPLZ77}(w, \varepsilon, \delta)|}{n} = \frac{|\mathtt{LZ77}(w)|}{n} + o(1).$$

Conclusion

- General framework to convert any compression scheme to **differentially private** compression scheme
  - Random amount of padding proportional to the **global sensitivity** of the compression scheme
- **Almost-tight upper and lower bound** for the global sensitivity of LZ77 compression
  - $\mathsf{GS}_{\mathrm{LZ77}}(n) = \mathcal{O}(n^{2/3}\log n)$ and $\mathsf{GS}_{\mathrm{LZ77}}(n) = \Omega(n^{2/3}\log^{1/3} n)$
  - With sliding window size W, $\mathsf{GS}_{\mathrm{LZ77}}(n) = \mathcal{O}(W^{2/3}\log n)$
  - Same upper bound holds for LZ77 with self-referencing (which allows overlapping matches)
- Since $\mathsf{GS}_{\mathrm{LZ77}}(n) = o(n)$, we can argue that **LZ77 is practical for DPCompress**:

$$\frac{|\mathtt{DPLZ77}(w,\varepsilon,\delta)|}{n} = \frac{|\mathtt{LZ77}(w)|}{n} + o(1).$$

Open Problems.

- Can we tighten/close the gap for LZ77?
- Global sensitivity for other compression schemes, e.g., Burrows-Wheeler Transform, Prefix-Free Parsing, etc.?
- "Sensitivity" for consecutive edits? (e.g., replace Alice's password with a random unrelated password)
- Can we design less sensitive compression schemes that are practical?

Conclusion

- General framework to convert any compression scheme to **differentially private** compression scheme
  - Random amount of padding proportional to the **global sensitivity** of the compression scheme
- **Almost-tight upper and lower bound** for the global sensitivity of LZ77 compression
  - $\mathrm{GS}_{\mathrm{LZ77}}(n) = \mathcal{O}(n^{2/3} \log n)$ and $\mathrm{GS}_{\mathrm{LZ77}}(n) = \Omega(n^{2/3} \log^{1/3} n)$
  - With sliding window size W, $\mathrm{GS}_{\mathrm{LZ77}}(n) = \mathcal{O}(W^{2/3} \log n)$
  - Same upper bound holds for LZ77 with self-referencing (which allows overlapping matches)
- Since $\mathrm{GS}_{\mathrm{LZ77}}(n) = o(n)$, we can argue that **LZ77 is practical for DPCompress**:

$$\frac{|\mathtt{DPLZ77}(w, \varepsilon, \delta)|}{n} = \frac{|\mathtt{LZ77}(w)|}{n} + o(1).$$

Open Problems.

- Can we tighten/close the gap for LZ77?
- Global sensitivity for other compression schemes, e.g., Burrows-Wheeler Transform, Prefix-Free Parsing, etc.?
- "Sensitivity" for consecutive edits? (e.g., replace Alice's password with a random unrelated password)
- Can we design less sensitive compression schemes that are practical?

Thank You!

References I

Tooru Akagi, Mitsuru Funakoshi, and Shunsuke Inenaga, **Sensitivity of string compressors and repetitiveness measures**, Inf. Comput. **291** (2023), no. C.

L. Peter Deutsch, **DEFLATE Compressed Data Format Specification version 1.3**, RFC 1951, May 1996.

Guillaume Lagarde and Sylvain Perifel, **Lempel-ziv: a "one-bit catastrophe" but not a tragedy**, Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, 2018, pp. 1478--1495.

Rafael Palacios, Andrea Fariña Fernández-Portillo, Eugenio F Sánchez-Úbeda, and Pablo García-De-Zúñiga, **Htb: a very effective method to protect web servers against breach attack to https**, IEEE Access **10** (2022), 40381--40390.