Balloon Hashing
A Memory-Hard Function with Provable Protection Against Sequential Attacks

Dan Boneh, Stanford
*Henry Corrigan-Gibbs, Stanford
Stuart Schechter, Microsoft Research
Balloon Hashing

A new password hashing function that:

1. Is **proven** memory-hard (in the sequential setting)

2. Uses a password-independent data access pattern

3. Matches the performance of the best heuristically secure memory-hard functions
# The Attacker’s Job

<table>
<thead>
<tr>
<th>User</th>
<th>Salt</th>
<th>H(passwd, salt)</th>
</tr>
</thead>
<tbody>
<tr>
<td>alice</td>
<td>0x65ff0162</td>
<td>0x526642d8</td>
</tr>
<tr>
<td>bob</td>
<td>0x37ceb328</td>
<td>0x5a325ad2</td>
</tr>
<tr>
<td>carol</td>
<td>0xec967ec1</td>
<td>0xf4441a71</td>
</tr>
<tr>
<td>dave</td>
<td>0xfb791a9a</td>
<td>0x1dbd71f3</td>
</tr>
</tbody>
</table>
The Attacker’s Job

<table>
<thead>
<tr>
<th>User</th>
<th>Salt</th>
<th>H(passwd, salt)</th>
</tr>
</thead>
<tbody>
<tr>
<td>alice</td>
<td>0x65ff0162</td>
<td>0x526642d8</td>
</tr>
<tr>
<td>bob</td>
<td>0x37ceb328</td>
<td>0x5a325ad2</td>
</tr>
<tr>
<td>carol</td>
<td>0xec967ec1</td>
<td>0xf4441a71</td>
</tr>
<tr>
<td>dave</td>
<td>0xfordb791a9a</td>
<td>0x1dbd71f3</td>
</tr>
</tbody>
</table>

For each row, attacker wants to make $2^{30}$ guesses.
Overall Goal

A good password hashing function makes the attacker’s job as difficult as possible.
Overall Goal

A good password hashing function makes the attacker’s job as difficult as possible.
Overall Goal

A good password hashing function makes the attacker’s job as difficult as possible.

If the authentication server can compute...

\[ X \text{ hashes per } \$ \text{ of energy} \]

then an attacker with custom hardware should only be able to compute...

\[ (1+\varepsilon)X \text{ hashes per } \$ \text{ of energy} \]
A good password hashing function makes the attacker’s job as difficult as possible.

If the authentication server can compute...

\[ X \text{ hashes per } \$ \text{ of energy} \]

then an attacker *with custom hardware* should only be able to compute...

\[ (1+\varepsilon)X \text{ hashes per } \$ \text{ of energy} \]

By this metric, conventional hash functions (e.g., SHA-256) are far from optimal!
SHA-256 Hashes (billions/$ of power)

Intel Westmere (Server) vs. Antminer S7 (Attacker)
SHA-256 Hashes (billions/$ of energy)

Intel Westmere (Server) vs Antminer S7 (Attacker)
SHA-256 Hashes (billions/$ of energy)

- Intel Westmere (Server)
- Antminer S7 (Attacker)
$512 on Amazon

SHA-256 Hashes (billions/$ of energy)

<table>
<thead>
<tr>
<th></th>
<th>Intel Westmere (Server)</th>
<th>Antminer S7 (Attacker)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA-256 Hashes</td>
<td>10</td>
<td>1,000,000</td>
</tr>
</tbody>
</table>

Intel Westmere (Server) and Antminer S7 (Attacker) with SHA-256 Hashes (billions/$ of energy).
SHA-256 Hashes (billions/$ of energy)

- **Intel Westmere (Server)**: 1
- **Antminer S7 (Attacker)**: 100,000,000

Comparing the processing power of the server and the attacker.
Intel Ivy Bridge-E Core i7-4960X
http://kylebennett.com/files/hfpics/IVB-E_%28LCC%29_Die_Wafer_Shot-7837.jpg
Memory Controller

Core
Core
Core
Core

I/O, Queue, etc.
Intel Ivy Bridge-E Core i7-4960X
http://kylebennett.com/files/hfpics/IVB-E_%28LCC%29_Die_Wafer_Shot-7837.jpg
Intel Ivy Bridge-E Core i7-4960X
http://kylebennett.com/files/hfpics/IVB-E_%28LCC%29_Die_Wafer_Shot-7837.jpg
Intel Ivy Bridge-E Core i7-4960X

http://kylebennett.com/files/hfpics/IVB-E_%28LCC%29_Die_Wafer_Shot-7837.jpg
Cost \approx \text{Area}
Cost \approx \text{Area}

1000000x efficiency gain!
Memory-Hardness
Memory-Hardness

**Memory-hard functions** use a large amount of working space during their computation

→ Attacker must keep caches on chip

→ Decreases the advantage of special-purpose HW

[Reinhold 1999], [Dwork, Goldberg, Naor 2003], [Abadi et al. 2005], [Percival 2009]
Memory-Hardness

**Memory-hard functions** use a large amount of working space during their computation

→ Attacker must keep caches on chip
→ Decreases the advantage of special-purpose HW

[Reinhold 1999], [Dwork, Goldberg, Naor 2003], [Abadi et al. 2005], [Percival 2009]

**Typical technique:**

1. **Fill** – fill buffer with pseudo-random bytes
2. **Mix** – read and write pseudo-random blocks in buffer
3. **Extract** – extract function output from buffer contents
Without memory-hardness
Plan

I. Background on password hashing
II. Goals
III. The Balloon algorithm
IV. Discussion
Plan

I. Background on password hashing
II. Goals
III. The Balloon algorithm
IV. Discussion
Plan

I. Background on password hashing

II. Goals

III. The Balloon algorithm

IV. Discussion
Goal 1: Memory-Hardness

Random oracles: [Bellare & Rogaway 1993]
Memory-hard functions: [Abadi et al. 2005] [Percival 2009]
Goal 1: Memory-Hardness

Informally, a memory-hard function, with hardness parameter N, requires space $S$ and time $T$ to compute, where

$$S \cdot T \in \Omega(N^2)$$

in the random-oracle model.

Random oracles: [Bellare & Rogaway 1993]
Memory-hard functions: [Abadi et al. 2005] [Percival 2009]
Goal 1: Memory-Hardness

Informally, a memory-hard function, with hardness parameter $N$, requires space $S$ and time $T$ to compute, where

$$S \cdot T \in \Omega(N^2)$$

in the random-oracle model.

**Intuition:** any adversary who tries to save *space* will pay a large penalty in computation *time*.

Random oracles: [Bellare & Rogaway 1993]
Memory-hard functions: [Abadi et al. 2005] [Percival 2009]
Goal 2: Side-Channel Resistance
Goal 2: Side-Channel Resistance

• The memory access pattern should not leak information about the password being hashed
  [Tsunoo et al. 2003] [Bernstein 2005] [Bonneau & Mironov 2006] […]
Goal 2: Side-Channel Resistance

- The memory access pattern should not leak information about the password being hashed
  [Tsunoo et al. 2003] [Bernstein 2005] [Bonneau & Mironov 2006] […]

Goal 3: Real-World Practical
Goal 2: Side-Channel Resistance

- The memory access pattern should not leak information about the password being hashed
  [Tsunoo et al. 2003] [Bernstein 2005] [Bonneau & Mironov 2006] [...]

Goal 3: Real-World Practical

- The hash should be able to support hundreds of logins per second while filling L2 cache (or more)
Existing Schemes
Existing Schemes

bcrypt, PBKDF2 [Provos & Mazières 1999], [Kaliski 2000]
Not memory hard
Existing Schemes

**bcrypt, PBKDF2** [Provos & Mazières 1999], [Kaliski 2000]
Not memory hard

**scrypt** [Percival 2009]
Password-dependent memory access pattern
Existing Schemes

bcrypt, PBKDF2 [Provos & Mazières 1999], [Kaliski 2000]
Not memory hard

scrypt [Percival 2009]
Password-dependent memory access pattern

Parallel-secure schemes [Alwen & Serbinenko 2015]
[Alwen, Blocki, Pietrzak 2016]
May be impractical for realistic parameter sizes
Existing Schemes

**bcrypt, PBKDF2** [Provos & Mazières 1999], [Kaliski 2000]
Not memory hard

**scrypt** [Percival 2009]
Password-dependent memory access pattern

**Parallel-secure schemes** [Alwen & Serbinenko 2015] [Alwen, Blocki, Pietrzak 2016]
May be impractical for realistic parameter sizes

**Argon2i and Catena** [Biryukov et al. 2015] [Forler et al. 2015]
Lack formal security analysis
Existing Schemes

bcrypt, PBKDF2 [Provos & Mazières 1999], [Kaliski 2000]
Not memory hard

scrypt [Percival 2009]
Password-dependent memory access pattern

Parallel-secure schemes
May be impractical for realistic parameter sizes

Argon2i and Catena [Biryukov et al. 2015] [Forler et al. 2015]
Lack formal security analysis

We demonstrate a practical attack against Argon2i
Plan

I. Background on password hashing
II. Goals
III. The Balloon algorithm
IV. Discussion
Plan

I. Background on password hashing
II. Goals
III. The Balloon algorithm
IV. Discussion
Balloon Hashing

A password hashing function that:

1. Is proven memory-hard (in the sequential setting)

2. Uses a password-independent data access pattern

3. Matches the performance of the best heuristically secure memory-hard functions
Balloon Hashing

A password hashing function that:

1. Is **proven** memory-hard (in the sequential setting)

2. Uses a password-independent data access pattern

3. Matches the performance of the best heuristically secure memory-hard functions
Balloon(password, salt, N = space_cost, R = num_rounds):

δ ← 3  // A security parameter.
var B₁, ..., Bₙ  // A buffer of N blocks.

// Step 1: Fill Buffer
B₁ ← Hash(password, salt)
for i = 2, ..., N:
    Bi ← Hash(Bi-1)

// Step 2: Mix Buffer
for r = 1, ..., R:
    for i = 1, ..., N:
        // Chosen pseudorandomly from salt
        (v₁, ..., vₜ) ← Hash(salt, r, i) ∈ Z_N^δ
        Bi ← Hash(B((i-1 mod N), Bi, Bᵥ₁, ..., Bᵥₜ)

// Step 3: Extract
return Bₙ
Balloon(password, salt, N = space_cost, R = num_rounds):

δ ← 3  // A security parameter.
var B₁, ..., Bₙ  // A buffer of N blocks.

// Step 1: Fill Buffer
B₁ ← Hash(password, salt)
for i = 2, ..., N:
    Bi ← Hash(Bᵢ₋₁)

// Step 2: Mix Buffer
for r = 1, ..., R:
    for i = 1, ..., N:
        // Chosen pseudorandomly from salt
        (v₁, ..., vₜ) ← Hash(salt, r, i) ∈ Zᴺδ
        Bi ← Hash(B(ᵢ₋₁ mod N), Bi, Bᵥ₁, ..., Bᵥₜ)

// Step 3: Extract
return Bₙ

A conventional hash function (e.g., SHA-256)
Balloon(password, salt, N = space_cost, R = num_rounds):
\[
\delta \leftarrow 3 \quad // A security parameter.
\]
\[
\text{var } B_1, \ldots, B_N \quad // A buffer of N blocks.
\]

// Step 1: Fill Buffer
\[
B_1 \leftarrow \text{Hash}(\text{password, salt})
\]
\[
\text{for } i = 2, \ldots, N:
B_i \leftarrow \text{Hash}(B_{i-1})
\]

// Step 2: Mix Buffer
\[
\text{for } r = 1, \ldots, R:
\]
\[
\text{for } i = 1, \ldots, N:
// Chosen pseudorandomly from salt
(v_1, \ldots, v_\delta) \leftarrow \text{Hash}(\text{salt, r, i}) \in \mathbb{Z}_N^\delta
B_i \leftarrow \text{Hash}(B_{(i-1 \mod N)}, B_i, B_{v_1}, \ldots, B_{v_\delta})
\]

// Step 3: Extract
\[
\text{return } B_N
\]
Balloon Hashing Algorithm
Balloon Hashing Algorithm

salt

passwd
Balloon Hashing Algorithm

salt

passwd
Balloon Hashing Algorithm

salt

passwd

Hash
Balloon Hashing Algorithm

- salt
- passwd

Hash
Balloon Hashing Algorithm

class Hash

B_1

salt

passwd
Balloon Hashing Algorithm
Balloon Hashing Algorithm

Hash

B₁  B₂  B₃
Balloon Hashing Algorithm
Balloon Hashing Algorithm
Balloon Hashing Algorithm
Balloon Hashing Algorithm
Balloon Hashing Algorithm

Hash

B_1  B_2  B_3  ...  B_N
Balloon Hashing Algorithm
Balloon Hashing Algorithm

Hash

B₁  B₂  B₃  ...  Bₙ
Balloon Hashing Algorithm
Balloon Hashing Algorithm

Hash

B_1  B_2  B_3  ...  B_N
Balloon Hashing Algorithm

Hash

B_1  B_2  B_3  ...  B_N
Balloon Hashing Algorithm
Balloon Hashing Algorithm
Balloon Hashing Algorithm

Hash

B₁  B₂  B₃  ...  Bₙ
Balloon Hashing Algorithm
Balloon Hashing Algorithm
Balloon Hashing Algorithm

B1  B2  B3  ...  BN
Balloon Hashing Algorithm
Balloon Hashing Algorithm

Hash

B_1  B_2  B_3  ...  B_N
Balloon Hashing Algorithm
Balloon Hashing Algorithm
Balloon Hashing Algorithm
Balloon Hashing Algorithm
Balloon Hashing Algorithm

B_1, B_2, B_3, ..., B_N
Balloon Hashing Algorithm
Balloon Hashing Algorithm
Balloon Hashing Algorithm
Balloon Hashing Algorithm

Hash

B_1  B_2  B_3  ...  B_N
Balloon Hashing Algorithm

Hash

$B_1 \quad B_2 \quad B_3 \quad \ldots \quad B_N$
Balloon Hashing Algorithm
Balloon Hashing Algorithm

A “mode of operation” for a cryptographic hash function
Balloon Hashing

A password hashing function that:

1. Is *proven* memory-hard (in the sequential setting)
2. Uses a password-independent data access pattern
3. Matches the performance of the best heuristically secure memory-hard functions
A password hashing function that:

1. Is proven memory-hard (in the sequential setting)

2. Uses a password-independent data access pattern

3. Matches the performance of the best heuristically secure memory-hard functions

Balloon Hashing
A password hashing function that:

1. Is **proven** memory-hard (in the sequential setting)

2. Uses a password-independent data access pattern

3. Matches the performance of the best heuristically secure memory-hard functions
A password hashing function that:

1. Is proven memory-hard (in the sequential setting)

2. Uses a password-independent data access pattern

3. Matches the performance of the best heuristically secure memory-hard functions
**Theorem** [informal]:
Computing the N-block R-round Balloon function w.h.p.,
when $\delta=7$, with space $S \leq N/8$ requires time $T$ such that

$$S \cdot T \geq \frac{(2^R - 1)}{32} \cdot N^2.$$
Theorem [informal]:
Computing the N-block R-round Balloon function w.h.p., when $\delta=7$, with space $S \leq N/8$ requires time $T$ such that

$$S \cdot T \geq \frac{(2^R - 1)}{32} \cdot N^2.$$ 

Saving a factor of 8 in space causes a slowdown exponential in # rounds.
Theorem [informal]:
Computing the N-block R-round Balloon function w.h.p., when $\delta=7$, with space $S \leq N/8$ requires time $T$ such that

$$S \cdot T \geq \frac{(2^R - 1)}{32} \cdot N^2.$$ 

When $R=20$, using $8\times$ less space requires using $15,000\times$ more time.
Theorem [informal]:
Computing the N-block R-round Balloon function w.h.p.,
when \( \delta=7 \), with space \( S \leq N/8 \) requires time \( T \) such that

\[
S \cdot T \geq \frac{(2^R - 1)}{32} \cdot N^2.
\]
Proving Memory-Hardness

**Theorem** [informal]:
Computing the N-block R-round Balloon function w.h.p., when $\delta=7$, with space $S \leq N/8$ requires time $T$ such that

$$S \cdot T \geq \frac{(2^R - 1)}{32} \cdot N^2 .$$

The proof works by inspecting the Balloon computation's data-dependency graph.

We draw heavily on prior work on pebbling arguments

[Paterson & Hewitt 1970] [Paul & Tarjan 1978] [Dwork, Naor, Wee 2005] [Dziembowski, Kazana, Wichs 2011] [Alwen & Serbinenko 2015]
Using Balloon ($\delta=3$). Both algorithms take four passes over memory.
Minimum buffer size required

Hashes/sec (one core)

PBKDF2

10^5 iters of SHA512

bcrypt

Using Balloon (δ=3). Both algorithms take four passes over memory.
Using Balloon (δ=3). Both algorithms take four passes over memory.
Using Balloon ($\delta=3$). Both algorithms take four passes over memory.
Using Balloon ($\delta=3$). Both algorithms take four passes over memory.
Using Balloon ($\delta=3$). Both algorithms take four passes over memory.
Using Balloon ($\delta=3$). Both algorithms take four passes over memory.
At 10 hashes per sec per core, how much memory can you fill?

Using Balloon ($\delta=3$). Both algorithms take four passes over memory.
At 10 hashes per sec per core, how much memory can you fill?

Using Balloon ($\delta=3$). Both algorithms take four passes over memory.
At 10 hashes per sec per core, how much memory can you fill?

Using Balloon ($\delta=3$). Both algorithms take four passes over memory.
At 10 hashes per sec per core, how much memory can you fill?

Using Balloon ($\delta=3$). Both algorithms take four passes over memory.
Using Balloon ($\delta=3$). Both algorithms take four passes over memory.
Minimum buffer size required

Hashes/sec (one core)

Using Balloon ($\delta=3$). Both algorithms take four passes over memory.
Using Balloon ($\delta=3$). Both algorithms take four passes over memory.
Using Balloon ($\delta=3$). Both algorithms take four passes over memory.
Using Balloon ($\delta=3$). Both algorithms take four passes over memory.
Using Balloon (δ=3). Both algorithms take four passes over memory.
Using Balloon ($\delta=3$). Both algorithms take four passes over memory.
Using Balloon ($\delta=3$). Both algorithms take four passes over memory.
Using Balloon ($\delta=3$). Both algorithms take four passes over memory.
Plan

I. Background on password hashing
II. Goals
III. The Balloon algorithm
IV. Discussion
Plan

I. Background on password hashing
II. Goals
III. The Balloon algorithm
IV. Discussion
Discussion: Parallel attacks
Discussion: Parallel attacks

- Alwen and Blocki (2016) show, in a parallel setting, it’s possible to execute a space-saving attack against any memory-hard function w/ data-indep access pattern
Discussion: Parallel attacks

• Alwen and Blocki (2016) show, in a parallel setting, it’s possible to execute a space-saving attack against any memory-hard function w/ data-indep access pattern
  – Including Balloon, Argon2i, etc.
Discussion: Parallel attacks

- Alwen and Blocki (2016) show, in a parallel setting, it’s possible to execute a space-saving attack against any memory-hard function with data-indep access pattern
  - Including Balloon, Argon2i, etc.

- The attack only applies when the memory usage is large enough (> 1 GB, but would only use ~16 MB in practice)
Discussion: Parallel attacks

• Alwen and Blocki (2016) show, in a parallel setting, it’s possible to execute a space-saving attack against any memory-hard function w/ data-indep access pattern
  – Including Balloon, Argon2i, etc.

• The attack only applies when the memory usage is large enough (> 1 GB, but would only use ~16 MB in practice)

• The attack would require special-purpose hardware with many cores and shared memory
Discussion: Parallel attacks

- Alwen and Blocki (2016) show, in a parallel setting, it’s possible to execute a space-saving attack against any memory-hard function w/ data-indep access pattern
  - Including Balloon, Argon2i, etc.

- The attack only applies when the memory usage is large enough (> 1 GB, but would only use ~16 MB in practice)

- The attack would require special-purpose hardware with many cores and shared memory

→ Not yet clear whether these attacks are of practical concern.
Discussion: Comparison with Argon2
Discussion: Comparison with Argon2

Argon2: Winner of the recent Password Hashing Competition
Discussion: Comparison with Argon2

Argon2: Winner of the recent Password Hashing Competition
  – Simple design, likely will see wide adoption
Discussion: Comparison with Argon2

Argon2: Winner of the recent Password Hashing Competition
- Simple design, likely will see wide adoption
- No proof of memory-hardness (in any model)
Discussion: Comparison with Argon2

Argon2: Winner of the recent Password Hashing Competition
- Simple design, likely will see wide adoption
- No proof of memory-hardness (in any model)
- Argon2i = variant with data-independent access pattern
Discussion: Comparison with Argon2

Argon2: Winner of the recent Password Hashing Competition
- Simple design, likely will see wide adoption
- No proof of memory-hardness (in any model)
- Argon2i = variant with data-independent access pattern

Our Contributions
Discussion: Comparison with Argon2

Argon2: Winner of the recent Password Hashing Competition
- Simple design, likely will see wide adoption
- No proof of memory-hardness (in any model)
- Argon2i = variant with data-independent access pattern

Our Contributions
• We demonstrate a practical attack against Argon2i’s memory-hardness properties
  (Designers have since modified the construction)
Discussion: Comparison with Argon2

Argon2: Winner of the recent Password Hashing Competition
- Simple design, likely will see wide adoption
- No proof of memory-hardness (in any model)
- Argon2i = variant with data-independent access pattern

Our Contributions
- We demonstrate a practical attack against Argon2i’s memory-hardness properties
  (Designers have since modified the construction)
- We prove that much better attacks are impossible
Discussion: Comparison with Argon2

Argon2: Winner of the recent Password Hashing Competition
- Simple design, likely will see wide adoption
- No proof of memory-hardness (in any model)
- Argon2i = variant with data-independent access pattern

Our Contributions
- We demonstrate a practical attack against Argon2i’s memory-hardness properties
  (Designers have since modified the construction)
- We prove that much better attacks are impossible

→ Balloon has stronger proven security properties than Argon2i.
  (In practice… )
Conclusion

Henry Corrigan-Gibbs
henrycg@stanford.edu

https://eprint.iacr.org/2016/027
https://github.com/henrycg/balloon/
Conclusion

• Memory-hard password hashing functions increase the cost of offline dictionary attacks.

Henry Corrigan-Gibbs
henrycg@stanford.edu

https://eprint.iacr.org/2016/027
https://github.com/henrycg/balloon/
Conclusion

• Memory-hard password hashing functions increase the cost of offline dictionary attacks.

• Balloon is a password hashing function that:
  – has proven memory-hardness properties against sequential attacks,
  – uses a password-indep. access pattern, and
  – is fast enough for real-world use.

Henry Corrigan-Gibbs
henrycg@stanford.edu

https://eprint.iacr.org/2016/027
https://github.com/henrycg/balloon/
Conclusion

- Memory-hard password hashing functions increase the cost of offline dictionary attacks.

- Balloon is a password hashing function that:
  - has proven memory-hardness properties against sequential attacks,
  - uses a password-indep. access pattern, and
  - is fast enough for real-world use.

- Balloon+SHA512* is strictly better than iterated hashing (PBKDF2-SHA512).

Henry Corrigan-Gibbs
henrycg@stanford.edu

https://eprint.iacr.org/2016/027
https://github.com/henrycg/balloon/
Conclusion

• Memory-hard password hashing functions increase the cost of offline dictionary attacks.

• Balloon is a password hashing function that:
  – has proven memory-hardness properties against sequential attacks,
  – uses a password-indep. access pattern, and
  – is fast enough for real-world use.

• Balloon+SHA512* is strictly better than iterated hashing (PBKDF2-SHA512).

https://eprint.iacr.org/2016/027
https://github.com/henrycg/balloon/

Henry Corrigan-Gibbs
henrycg@stanford.edu
Conclusion

- Memory-hard password hashing functions increase the cost of offline dictionary attacks.
- Balloon is a password hashing function that:
  - has proven memory-hardness properties against sequential attacks,
  - uses a password-indep. access pattern, and
  - is fast enough for real-world use.
- Balloon+SHA512* is strictly better than iterated hashing (PBKDF2-SHA512).

Henry Corrigan-Gibbs
henrycg@stanford.edu