A Future-Adaptable Password Scheme

Niels Provos and David Mazières

{provos, dm}@openbsd.org

The OpenBSD Project
Overview

1. Introduction
2. Design Criteria
3. Eksblowfish
4. Bcrypt
5. Comparison and Evaluation
6. Conclusion
Introduction

As computers grow faster

- cryptography becomes feasible to make systems more secure
- attacks get more powerful
- quality of user-chosen passwords remains the same.

Security protocols still depend on secret passwords but fail to adapt algorithms to more resourceful attackers.
**Introduction**

UNIX has *crypt* as password hash for authentication

- over twenty years old, still in use
- hash inversion only by password guessing
- 1976 4 crypt/second, today > 200,000 crypt/second
Introduction

Two types of password guessing attacks

- on-line
  - easy to defend against, just slow down
- off-line
  - out of server control, only defence is computational cost

Most research centered around communication over insecure networks

- Encrypted Key Exchange (EKE)
- Secure Remote Password Protocol (SRP)

But if secret server state is known \( \Rightarrow \) off-line password guessing
Password scheme

- “as good as the passwords users choose”

\[ \forall D, \forall P, \forall A, \]
\[ \left| \Pr[t_1 \leftarrow T, \ldots, t_c \leftarrow T, s \leftarrow D, \right. \right. \]
\[ b \leftarrow A(t_1, F(s, t_1), \ldots, t_c, F(s, t_c)); \]
\[ b = P(s) \right] \]
\[ - \Pr[t_1 \leftarrow T, \ldots, t_c \leftarrow T, s \leftarrow D, \]
\[ b \leftarrow A(t_1, F(s, t_1), \ldots, t_c, F(s, t_c)), \]
\[ s' \leftarrow D; b = P(s') \right] \]
\[ < \frac{\epsilon}{2} \cdot |A| \cdot R(D) \]

\( D \) probability distribution on passwords \( A \) attacker as randomized boolean circuit
\( P \) predicate about a password \( T \) probability distribution
Design Criteria

Password scheme

- makes non-trivial use of all inputs, 2nd pre-image resistant

\[\forall D, \forall A,\]
\[\Pr[t \leftarrow T, s \leftarrow D, s' \leftarrow A(s, t);\]
\[s \neq s' \land F(s, t) = F(s', t)]\]
\[< \epsilon \cdot |A| \cdot R(D)\]

\(D\) probability distribution on passwords \(T\) probability distribution
\(A\) attacker as randomized boolean circuit
Design Criteria

For a good password scheme, the properties imply

- a *salt* as 2nd input to thwart lookup tables

- increasing evaluation cost over time

- efficient algorithm ⇒ no speedup in hardware
Eksblowfish

Expensive key schedule blowfish

- cost parameterizable and salted
- takes user-chosen passwords as keys
- based on blowfish block cipher by Schneier
Eksblowfish

Encryption identical to blowfish, key setup differs

EksBlowfishSetup \((cost, salt, key)\)

\[\begin{align*}
state & \leftarrow \text{InitState} () \\
state & \leftarrow \text{ExpandKey}(state, salt, key) \\
& \textbf{repeat} \ (2^{cost}) \\
& \quad state \leftarrow \text{ExpandKey}(state, 0, salt) \\
& \quad state \leftarrow \text{ExpandKey}(state, 0, key) \\
& \textbf{return} \ state
\end{align*}\]

Changing state

- might make unknown optimizations less applicable
- requires 4KB of constantly accessed and modified memory.
- limits usefulness of hardware pipelining
Bcrypt is a password scheme that

- uses eksblowfish
- has 128-bit salt and encrypts 192-bit magic value
- fulfills properties: 2nd pre-image resistant, cost adaptable, large salt space

```
bcrypt (cost, salt, pwd)
    state ← EksBlowfishSetup (cost, salt, key)
    ctext ← “OrpheanBeholderScryDoubt”
    repeat (64)
        ctext ← EncryptECB (state, ctext)
    return Concatenate (cost, salt, ctext)
```
Bcrypt

Implemented and deployed since OpenBSD 2.1

- random numbers generated via \textit{arc4random(3)} using kernel entropy pool.

- choice of password scheme with \textit{passwd.conf}

Version identifier and cost encoded in hash

$2a$08$U32pv8knIHG4coal9sMab0hkiNj0mfTZFbwV8axMIfno8/x5zMD
Comparison and Evaluation

Comparison with two widespread hashing functions

- traditional crypt, 56-bit DES key from password
  - restricted password size
  - too small salt space
  - constant execution cost

- MD5 crypt by Poul-Henning Kamp
  - virtually no limit on password length but only 128-bit output
  - constant execution cost
Comparison and Evaluation

Dictionary attack

- users pick predictable passwords
- hash dictionary words and compare

_Bcrypt_ can be made so slow that dictionary attack is impractical

Salt collisions

- optimize dictionary attack by grouping together passwords with same salt
- small salts, poor random numbers ⇒ more collisions

_Bcrypt’s_ 128-bit salt virtually guarantees uniqueness
Comparison and Evaluation

Precomputing Dictionaries

- precompute hashes of common passwords for all salts
- store in database $\Rightarrow$ computing hash = database lookup

*bcrypt*’s salt space too large to store, also no other precomputation possible
Comparison and Evaluation

Algorithm Optimization

Attack more practical with lower computational cost

- DES crypt 5× faster with Biham’s bitslicing on Alpha
- “John the Ripper” considerable speedup of MD5 crypt

Bcrypt can not be bitsliced because S-Boxes change
Comparison and Evaluation

Hardware Improvements

- > 200,000 crack/second on modern Alpha for traditional crypt
- specialized hardware like EFF DES cracker

MD5 and traditional crypt use fixed number of rounds ⇒ become easier to compute with time.

Bcrypt

- adapts to faster processors
- uses only simple operations ⇒ no advantage with specialized hardware
Conclusion

- quality of user-chosen passwords remains fixed with time
- traditional password schemes fail to adapt to more powerful attackers
- “as good as the passwords users choose” ⇒ eksblowfish and bcrypt
- bcrypt can
  - replace UNIX password hash, as done in OpenBSD
  - enhance security of other protocols