Note:

These slides reflect the viewpoint of some crypto engineers at Microsoft at a specific point in time for the purpose of open discussion of the PHC work-in-progress. It should not be interpreted as commitment by Microsoft to implement any particular functionality, indicate future product direction, etc. If you have any questions, please don’t hesitate to contact the authors at the email addresses listed at the end.

Thanks! ☺

What Microsoft Would like from the Password Hashing Competition

Marsh Ray, Microsoft Azure

Greg Zaverucha, Microsoft Research

Microsoft

It’s complicated…

We found the same questions that are raised in the industry at large

You guys are not being clear about the threat model or scenario […] changing your position …

I'm not really sure that's worth the time

If it's easy to implement and cheap from a CPU perspective, then I don't think there will be controversy.

This is why we need to get everyone to multi-factor authentication. Iterative hashing is a legacy defense-in-depth measure with marginal value.

If someone built specialized hardware, and you got say 3 orders of magnitude speed up …

- A common response was also
  - *Chirping crickets*

Microsoft

Why does the PHC exist?

- Isn’t salted hashing enough?
- Isn’t PBKDF2-SHA-1 good enough?
- How will this affect application response times?
- Why don’t you just use two-factor auth?
- Why don’t you just encrypt the hashes?
- Why don’t you just put the hashes in an HSM?
- If the password file is compromised, so is user data, why does hashing help?

Microsoft

Why does the PHC exist?

- Isn’t salted hashing enough?
- Isn’t PBKDF2-SHA-1 good enough?
- How will this affect application response times?
- Why don’t you just use two-factor auth?
- Why don’t you just encrypt the hashes?
- Why don’t you just put the hashes in an HSM?
- If the password file is compromised, so is user data, why does hashing help?

Hard benchmark data

Threat modeling

Microsoft

When does password hashing help ?

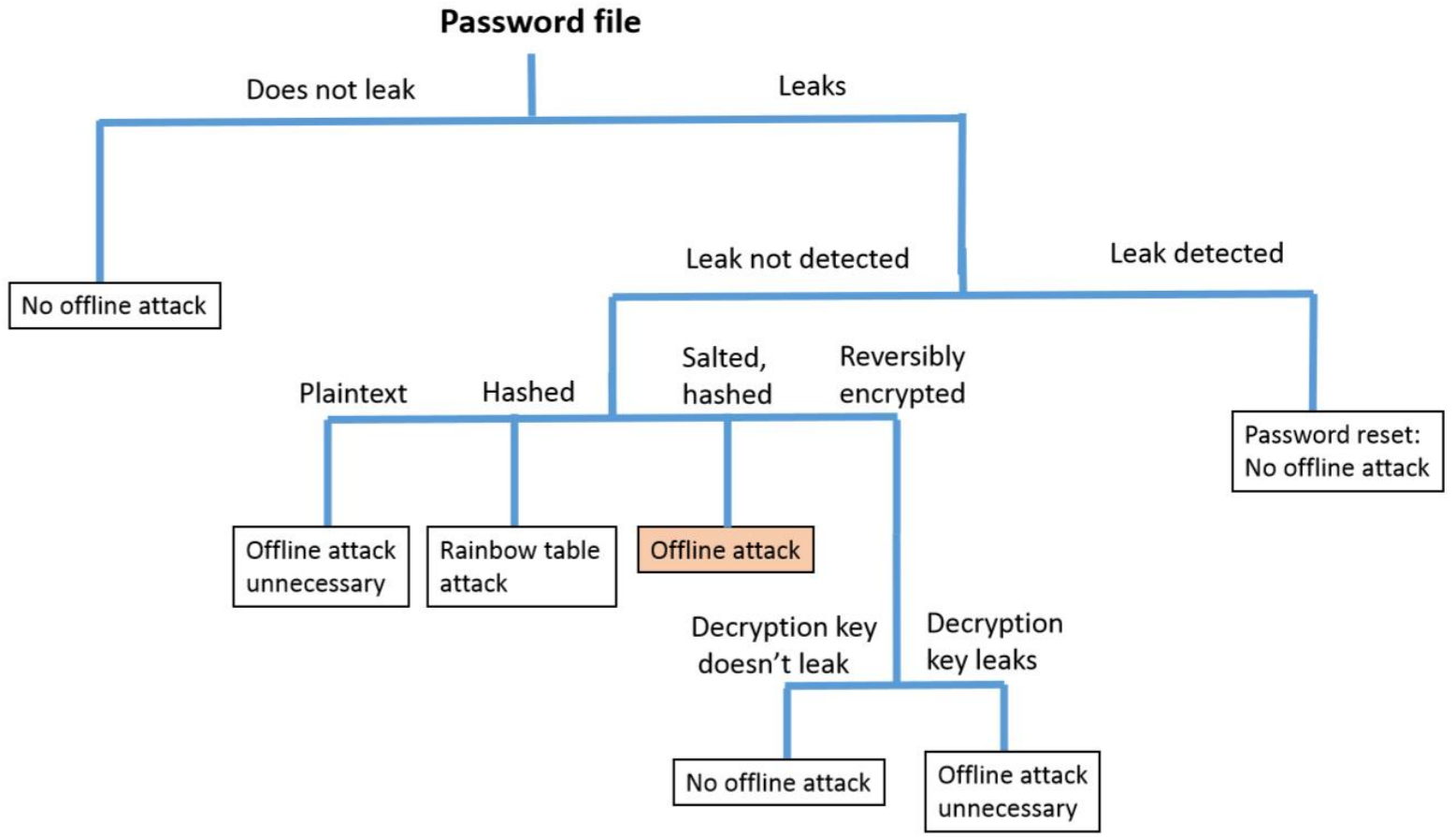


Fig. 1. Decision tree indicating the applicable threats depending on how the password file is handled. Observe that offline guessing is only a threat when the password file leaks, that fact goes undetected, and the passwords have been properly salted and hashed. In all other cases an offline guessing attack is either not necessary or not possible.

From
"An Administrator's Guide to Internet Password Research," Florencio, Herley and van Oorschot, Usenix LISA 2014 (to appear).

Why are more iterations better ?

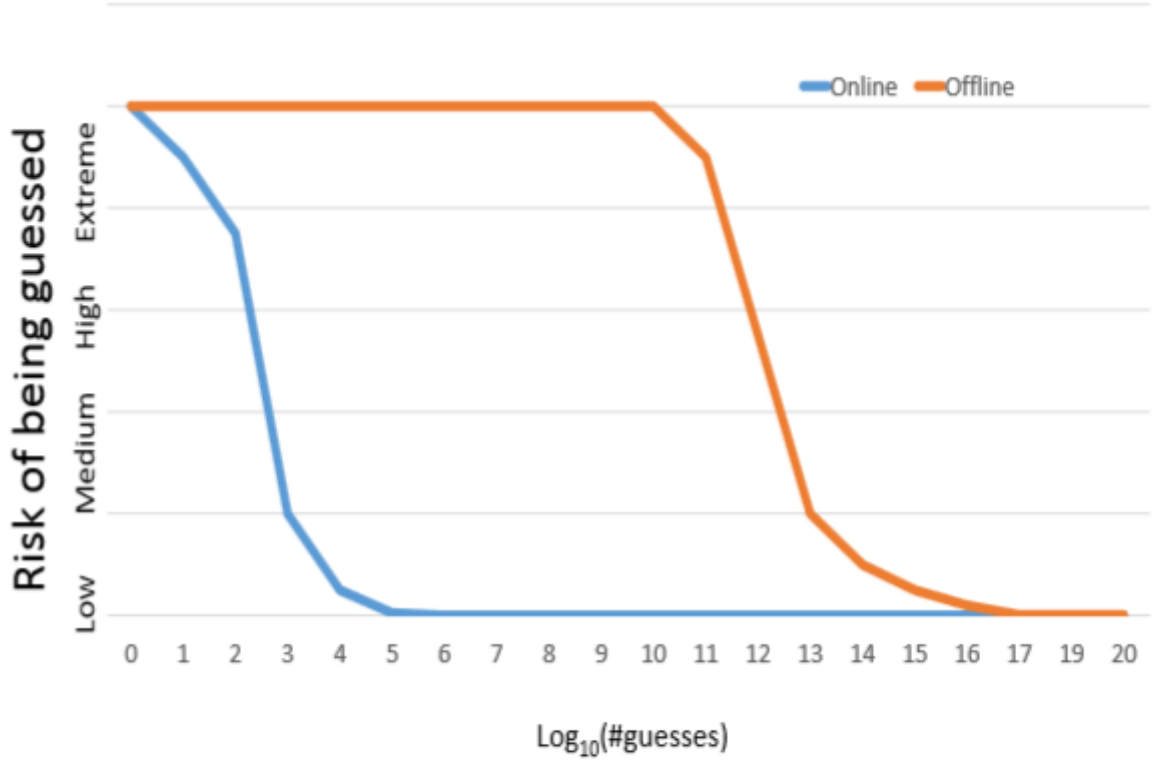
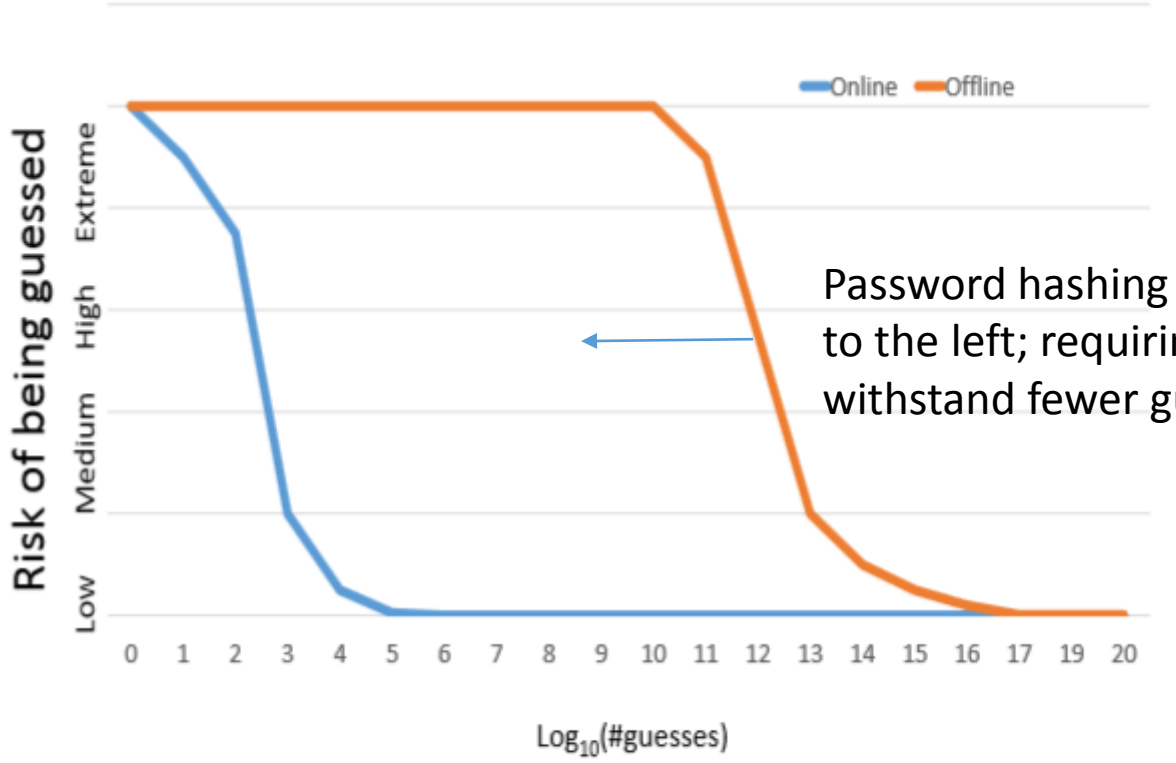


Fig. 2. Conceptualization of risk from online and offline guessing as a function of $\log_{10}$ of the number of guesses a password will withstand. Observe that, in the region between 10^6 and 10^{12}, improvement in the resistance to guessing has negligible effect on either online or offline guessing.

From
"An Administrator's Guide to Internet Password Research," Florencio, Herley and van Oorschot, Usenix LISA 2014 (to appear).

Why are more iterations better ?



Password hashing can shift this line to the left; requiring passwords to withstand fewer guesses

Fig. 2. Conceptualization of risk from online and offline guessing as a function of $\log_{10}$ of the number of guesses a password will withstand. Observe that, in the region between 10^6 and 10^{12}, improvement in the resistance to guessing has negligible effect on either online or offline guessing.

From
"An Administrator's Guide to Internet Password Research," Florencio, Herley and van Oorschot, Usenix LISA 2014 (to appear).

Outline

- What Microsoft thinks it wants
- What Microsoft needs from the PHC
  - Background, perspective
  - Use cases of interest
  - Requirements
  - Comparison of candidates based on requirements
- Where more work is needed
  - Security analysis
  - Parameter selection
  - Value proposition
    - For cryptographers
    - For app developers and server admins

Microsoft

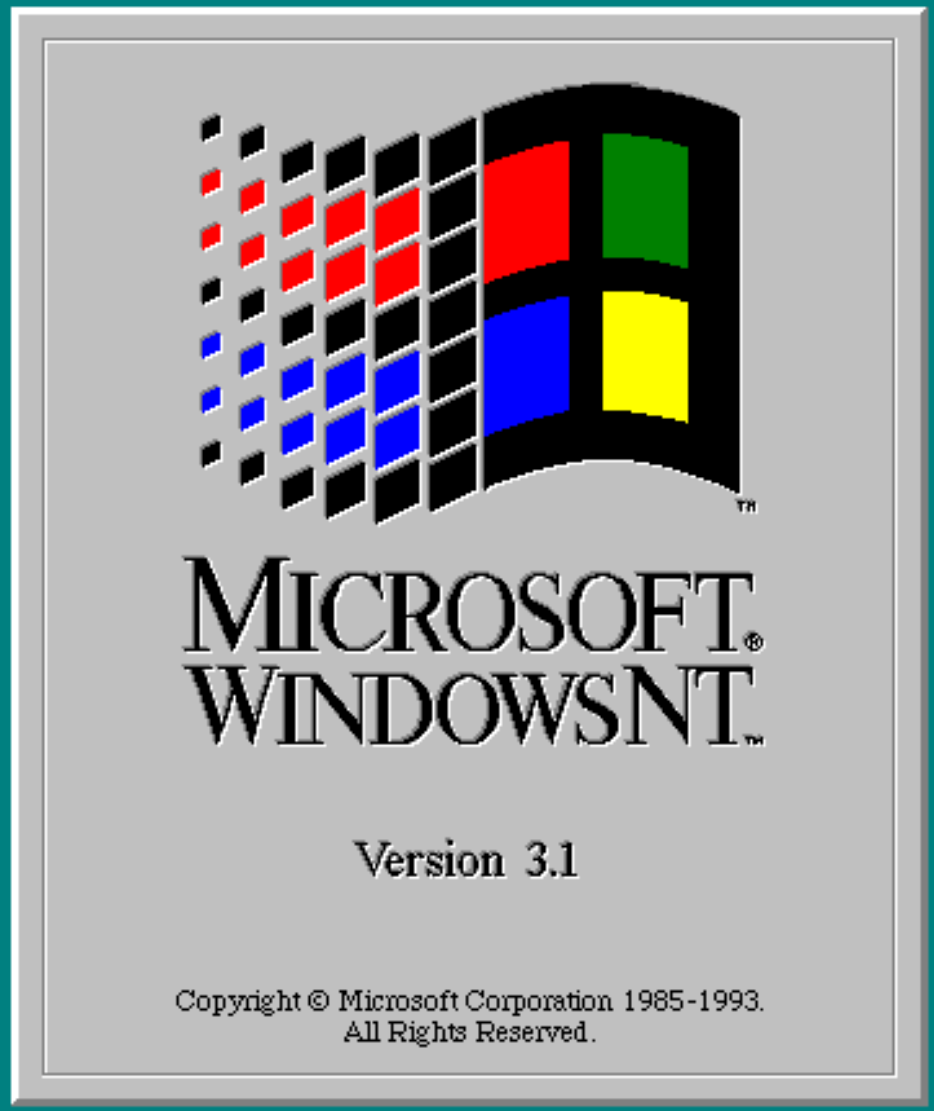
Background

- Passwords are used in many Microsoft products
  - Online services (Hotmail/Outlook.com, Xbox Live, Skype)
  - Device login (Windows, Windows Phone, Xbox)
  - File encryption (Office)
- Many other products, and parts of our infrastructure
- Presenters participate in design and security review of cryptographic features (across the whole company)
- We don’t have a design in the competition

Microsoft

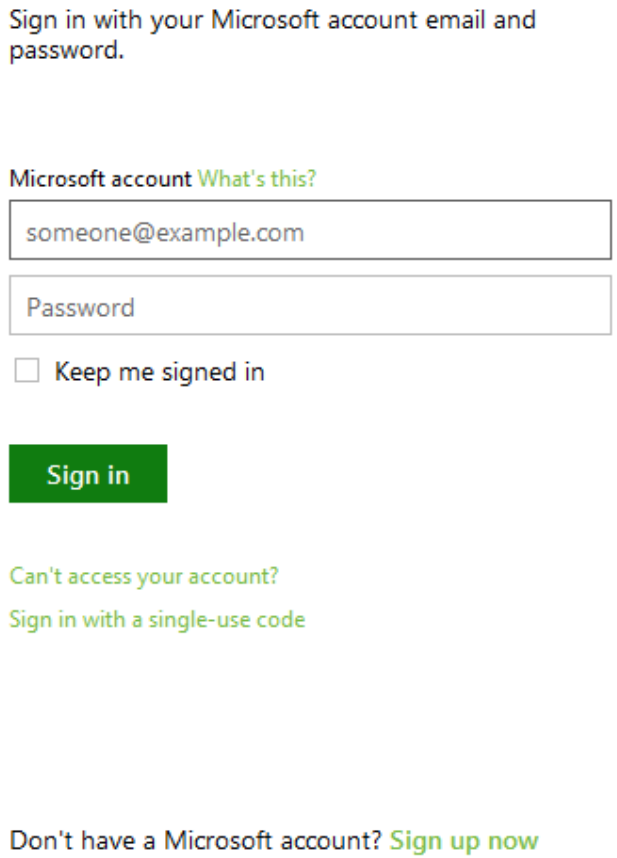
Trivia

- Which was the first version of Windows to use passwords for user accounts?


MICROSOFT®
WINDOWS NT™
Version 3.1
Copyright © Microsoft Corporation 1985-1993.
All Rights Reserved.

Use Case 1 – Online Services

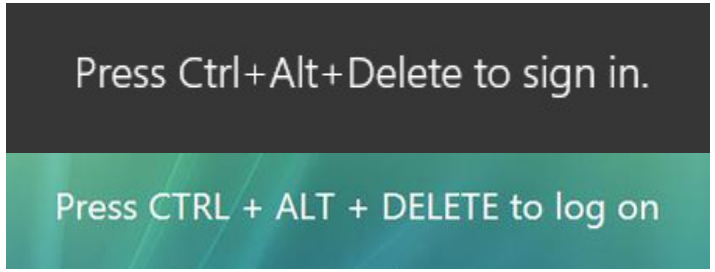
- Passwords used for remote authentication
- Multiple authentication servers, each handles 100s of requests/second
- Includes browser-based logins, as well as network services like file and print
- Page load times drive performance requirements
- More logins from apps (e.g., an app using OneDrive for storage)



Microsoft

Use Case 2 – Log on to a Device

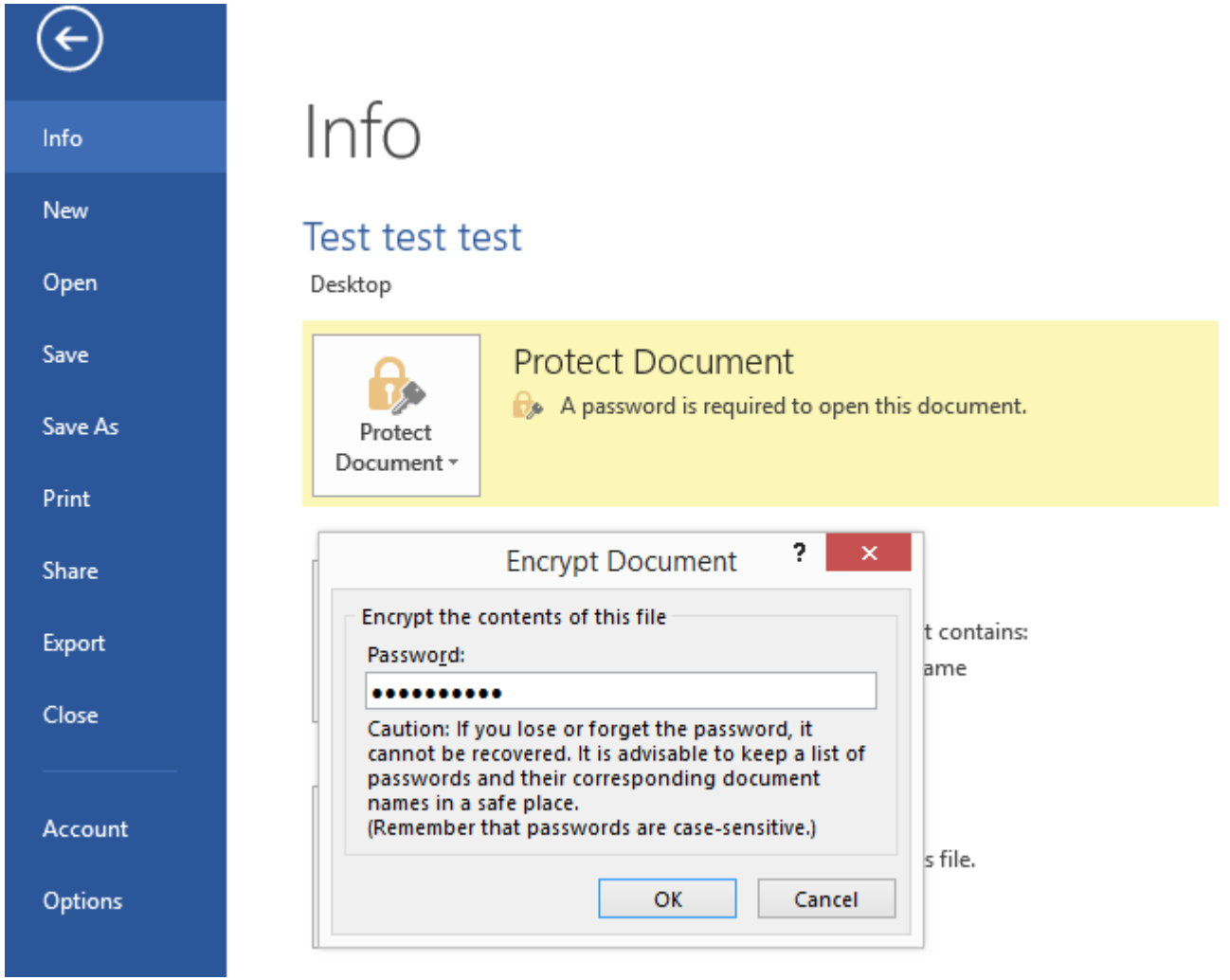
- The original use case for password hashing
- Windows systems (PC/laptop/servers)
- More cycles available for hashing (than online services), but still finite, responsiveness is important
- Wide variety of hardware



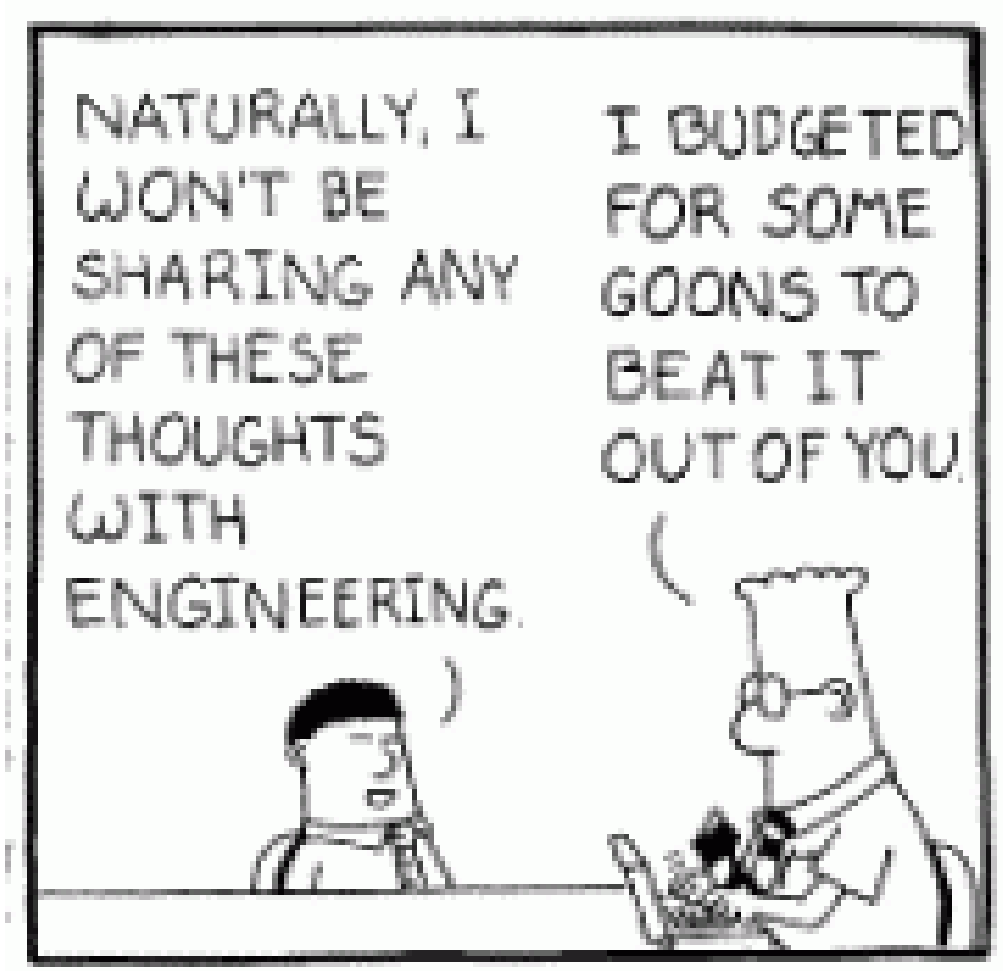
Microsoft

Use Case 3 – Password-Based Encryption

- Derive keys from the password to encrypt and authenticate data
- Used when key management is infeasible; lack of hardware or platform support. E.g., sending an encrypted Word document as an email attachment.
- Possibly the worse case scenario
  - The attacker is assumed to get the salt, ciphertext (and auth tag)
  - An attack in the other use cases is normal operation here
- Probably the most resources available for hashing



Requirements



Requirements

- In addition to the PHC Call for Submissions
- Not “deep” technical requirements (save these for round 2!)
- As users of the PHC winner, what features would we want it to have?
- Our requirements are derived from:
  - Well-understood design trade-offs
  - Use cases & our product mix

One “Function”

- Rather than one-per-use-case
- Easier to give guidance to product teams
- Easier to use and deploy
- Hard enough to get one function adopted in any large organization

Microsoft

Feature parity with PBDKF2

- To facilitate replacing existing uses of PBKDF2
  - It's the current standard (PKCS#5, RFC 2898)
  - Similar to AES v. DES, SHA3 v. SHA2
- "Agility" between PBKDF2 and PHC winner
- In particular
  - Have at least the same security guarantees
  - Secure as a key derivation function (KDF)
  - Same parameters, including variable-length inputs and outputs
  - *Ability* to use very little memory
- Bug parity is not desired

Microsoft

Migration guidance

- Clear path for engineers
- Mapping of parameter values from PBKDF2
  - Work factor
  - Algorithm choices

Microsoft

Perform well on common architectures

- Our use cases span x86, x64 and ARM(32/64)
- Design should not be “overfit” to processor features not shared by all
  - E.g. (simple) SIMD operations are available everywhere, but AES-NI is not
- Simplifies parameter selection across different hardware
- Defender might have special hardware
  - Attacker will, at his option

Microsoft

Perform well on common architectures

- Good approach is to make black-box use of other primitives, already optimized for the underlying hardware
  - No new ASM required
- Using existing primitives can also allow fast implementation in high-level languages
  - E.g., Javascript + web crypto API
  - E.g., C# + PInvoke platform implementations

Microsoft

Defined API for developers

- Compatible with C and high-level languages
- As fool-resistant as practical
- Could be two functions
  - phc_derive(pw, [optional parameters]) - > string
  - phc_verify(string, string) -> bool
- Result string encodes metadata, salt, etc.
- Provide good, conservative defaults to parameters
  - But defaults not set in stone, they must evolve over time

Microsoft

Resist Side Channel Attacks

- Resisting side channel attacks is important
- Broad range of use-cases and systems
  - From phones to VMs on shared hardware
- Crypto functions are often long-lived
  - PBKDF2 is old enough to drink (PKCS #5 published 1993)
- Two large classes of side channel leaks
  1. Timing attacks – time required varies with inputs. Most candidates can be implemented in constant time
  2. Cache timing attacks – victim and attacker share a cache, leaks information about victim memory accesses. More problematic for PHC candidates
- Typical countermeasures incur work for the defender, but not the attacker.
  - E.g., protected table look-ups

Microsoft

Resist Side Channel Attacks

Cache timing attacks

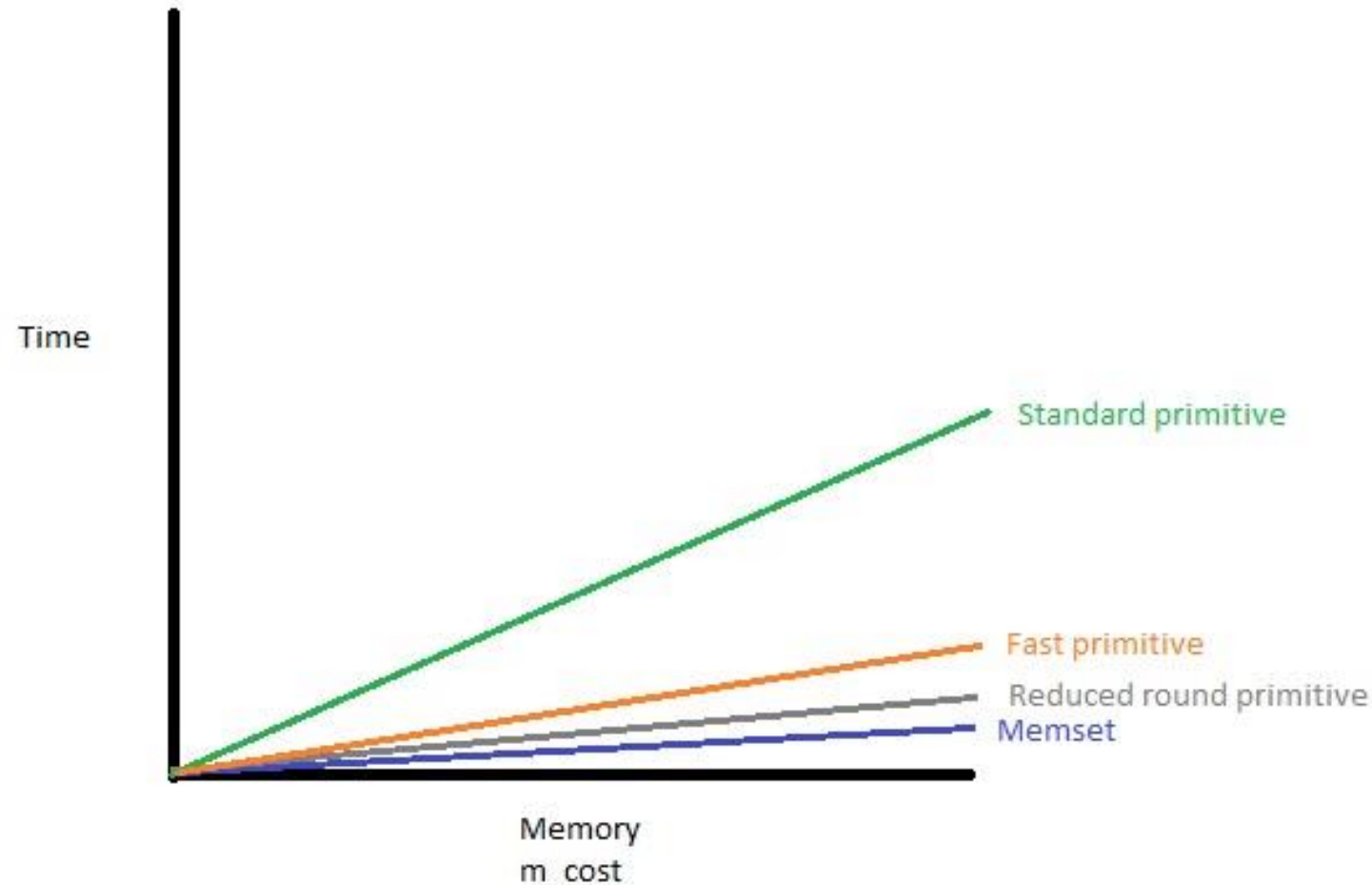
- In PHC candidates, the memory access pattern either
  1. Is independent of the password
  2. Depends on the password
  3. Depends on the password, but only after some amount of work
- Approach 1 has a fixed access pattern which allows attacker optimizations.
- Approach 2 uses the entropy of the password to make the access pattern unpredictable but opens a timing channel.
- Approach 3 attempts to make the trade-off.

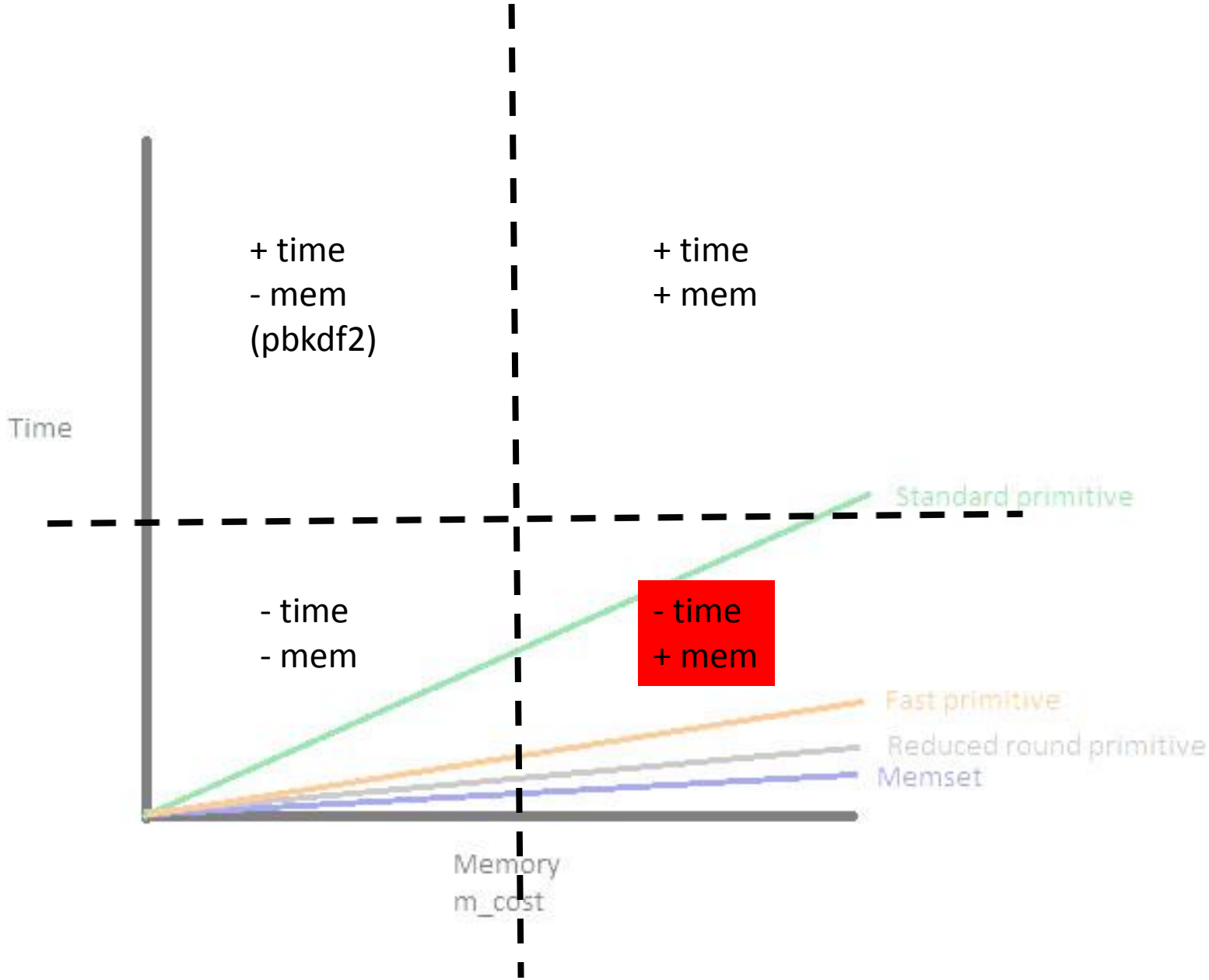
Microsoft

Agile in the choice of primitives

- Support agility in any use of standard primitive types
  - Flexibility is a useful feature of HMAC and PBKDF2
  - E.g., moving from HMAC-MD5 to HMAC-SHA256
  - E.g., moving from PBKDF2-HMAC-MD5 to PBDKF2-HMAC-SHA512
- Use of well-studied primitives can simplify formal security analysis
  - Reasonable assumptions can be made
  - e.g., SHA-2-512/256 can be modeled as a random oracle
- Use of reduced-round variants should be optional
  - I.e., don't *require* the use of weakened primitives
  - Many candidates use reduced-round primitives in order to fill memory fast

Microsoft


Time
Standard primitive
Fast primitive
Reduced round primitive
Memset
Memory
m_cost


+ time
- mem
(pbkdf2)
+ time
+ mem
Time
Standard primitive
- time
- mem
- time
+ mem
Fast primitive
Reduced round primitive
Memset
Memory
m_cost

Not *require* the use of weak algorithms

- *Allows conservative choice*: Applications that require (time-, mem+) can use the weak algorithm if necessary (they may use the strongest algorithm that allows their parameter choice)
- Aside: the security properties required by the reduced round primitive are often vague

Microsoft

But is this perhaps false agility?

- Has there ever been a case where primitive agility in a password hash function was really useful?
  - Unlike guts-for-speed AES or SHA, the work factor requirement gives absurdly generous safety margin
- Example: transition PBKDF2<SHA-1> to PBKDF2<SHA-512>
  - Really it just needed a proper memory-hardness parameter
- Makes analysis difficult, compared to defined tuning parameters
  - Is PHC<SHA-2-256> different from PHC<SHA-3-256> ?
- Let’s not push today’s uncertainties onto developers
  - We want the PHC to provide an expert-opinionated solution

Microsoft

Do we even want standard primitives?

- Most, or possibly all, modern standard primitives are intentionally designed to be optimal on custom hardware
  - The opposite of what a password hashing function wants!
- PHC is developing a different, newly-identified, type of primitive
  - Let’s not excessively constrain it!

- So if PHC doesn’t require standard primitives, do we need underlying primitive agility?

Microsoft

Nice to haves

- Standard digest format
- Reversibly encode inputs, to avoid things like
  HMAC(pw, salt) == HMAC(pw||0…0, salt) 

- Clear statement of security level as a KDF
- Can be used to add work factor to PAKEs
- Parameter selection guidance*
- Formal security analysis*

Microsoft

PHC Candidates

Versus these requirements

Not a message from the PHC Panel!

Comparison

	KDF function	KDF flexibility	CPU Arch. Neutral	Agility	Side ch. Resist.	Support all Use Cases
AntCrypt v0	Yes	No	Maybe	Yes	No	No
Argon v1	Yes	No	No	Maybe	No	No
battcrypt v0	Yes	Yes	Yes	No	No	Yes
Catena v1	Yes	Yes	Yes	Yes	Yes	Yes
Centrifuge v0	Yes	Yes	Yes	Yes	No	Yes
EARWORM v0	No	No	No	No	No	No
Gambit v1	Yes	Yes	Yes	Yes	Yes	Yes
Lanarea v0	Yes	Yes	Yes	Yes	No	Yes
Lyra2 v1	Yes	Yes	Yes	Yes	Maybe	Yes
Makwa v0	Yes	Yes	Yes	Yes	Yes	Yes
MCS_PHS v1	Yes	No	Yes	Yes	Yes	No
Omega Crypt v0	Yes	No	Yes	Maybe	No	No
Parallel v0	Yes	Yes	Maybe	Yes	Yes	No
PolyPassHash v0	No	No	Yes	Yes	Yes	No
POMELO v1	Maybe	No	Yes	No	No	No
Pufferfish v0	Yes	Yes	Yes	Maybe	Yes	Yes
RIG v1	Yes	Yes	Yes	Yes	Yes	Yes
Schvrch v0	Maybe	Maybe	Yes	No	No	Maybe
Tortuga v0	Maybe	Yes	Yes	No	Maybe	Maybe
TwoCats v0	Yes	No	Yes	Yes	Maybe	No
Yarn v2	Yes	No	Maybe	Maybe	No	No
yescrypt v0	Yes	Yes	Yes	Yes	No	Yes

Summary

- Clearly meeting all requirements: Catena, Gambit, Makwa, RIG.
- Potentially meeting all requirements (one or more Maybe): Lyra2 and Pufferfish.
- Missing only the side-channel requirement: Lanarea, yescrypt
- Meeting the use case requirement (but not all/potentially all requirements): battcrypt, Centrifuge

- Tweaks could change this list

Microsoft

Areas that need more work

More difficult nice-to-haves

For the PHC community at large, not just designers

Parameter Selection Guidance

- Ideally we'd have something like SP 800-57 part 1

Table 4: Security-strength time frames

Security Strength		2011 through 2013	2014 through 2030	2031 and Beyond
80	Applying	Deprecated	Disallowed	
	Processing	Legacy use		
112	Applying	Acceptable	Acceptable	Disallowed
	Processing			Legacy use
128	Applying/Processing	Acceptable	Acceptable	Acceptable
192		Acceptable	Acceptable	Acceptable
256		Acceptable	Acceptable	Acceptable

E.g., RSA 2048 OK for use until 2030, if you need security beyond 2030, use RSA 4096 or above

Microsoft

Key Lengths

Contribution to The Handbook of Information Security

Arjen K. Lenstra

Citibank, N.A., and Technische Universiteit Eindhoven
1 North Gate Road, Mendham, NJ 07945-3104, U.S.A.
arjen.lenstra@citigroup.com

Abstract. The key length used for a cryptographic protocol determines the highest security it can offer. If the key is found or 'broken', the security is undermined. Thus, key lengths must be chosen in accordance with the desired security. In practice, key lengths are mostly determined by standards, legacy system compatibility issues, and vendors. From a theoretical point of view selecting key lengths is more involved. Understanding the relation between security and key lengths and the impact of anticipated and unexpected cryptanalytic progress, requires insight into the design of the cryptographic methods and the mathematics involved in the attempts at breaking them. In this chapter practical and theoretical aspects of key size selection are discussed.

1 Introduction

In cryptographic context, 40, 56, 64, 80, 90, 112, 128, 155, 160, 192, 256, 384, 512, 768, 1024, 1536, 2048, and 4096 are examples of key lengths. What they mean and how they are and should be selected is the subject of this chapter.

Selecting Cryptographic Key Sizes

Arjen K. Lenstra
Arjen.Lenstra@citicorp.com

Eric R. Verheul
Eric.Verheul@nl.pwcglobal.com

November 15, 1999

Abstract. In this article we offer guidelines for the determination of key sizes for symmetric cryptosystems, RSA, and discrete logarithm based cryptosystems both over finite fields and over groups of elliptic curves over prime fields. Our recommendations are based on a set of explicitly formulated hypotheses, combined with existing data points about the cryptosystems.

Key Lengths

Contribution to The Handbook of Information Security

Arjen K. Lenstra

Citibank, N.A., and Technische Universiteit Eindhoven
1 North Gate Road, Mendham, NJ 07945-3104, U.S.A.
arjen.lenstra@citigroup.com

Abstract. The key length used for a cryptographic protocol determines the highest security it can offer. If the key is found or 'broken', the security is undermined. Thus, key lengths must be chosen in accordance with the desired security. In practice, key lengths are mostly determined by standards, legacy system compatibility issues, and vendors. From a theoretical point of view selecting key lengths is more involved. Understanding the relation between security and key lengths and the impact of anticipated and unexpected cryptanalytic progress, requires insight into the design of the cryptographic methods and the mathematics involved in the attempts at breaking them. In this chapter practical and theoretical aspects of key size selection are discussed.

1 Introduction

In cryptographic context, 40, 56, 64, 80, 90, 112, 128, 155, 160, 192, 256, 384, 512, 768, 1024, 1536, 2048, and 4096 are examples of key lengths. What they mean and how they are and should be selected is the subject of this chapter.

Selecting Cryptographic Key Sizes

Arjen K. Lenstra
Arjen.Lenstra@citicorp.com

Eric R. Verheul
Eric.Verheul@nl.pwcglobal.com

November 15, 1999

Abstract. In this article we offer guidelines for the determination of key sizes for symmetric cryptosystems, RSA, and discrete logarithm based cryptosystems both over finite fields and over groups of elliptic curves over prime fields. Our recommendations are based on a set of explicitly formulated hypotheses, combined with existing data points about the cryptosystems.

Existing Data Points

hashcat
advanced password recovery

hashcat
oclHashcat
oclGaussCrack
Forum
Wiki
Trac
Tools
Events
Converter
Contact

Tested GPU

- All CUDA and Stream enabled cards should work

Performance

- PC1: Windows 7, 64 bit · Catalyst 14.4 · 1x AMD hd7970 · stock core clock
- PC2: Windows 7, 32 bit · ForceWare 331.67 · 1x NVidia gtx580 · stock core clock
- PC3: Ubuntu 14.04, 64 bit · Catalyst 14.4-rev2 · 1x AMD hd6990 · stock core clock
- PC4: Ubuntu 14.04, 64 bit · ForceWare 331.67 · 1x NVidia gtx750Ti · stock core clock
- PC5: Ubuntu 14.04, 64 bit · Catalyst 14.4-rev2 · 8x AMD R9 290X · stock core clock

Hash Type	PC1	PC2	PC3		
MD4	15445M c/s	4245M c/s	19868M c/s		
MD5	7893M c/s	2802M c/s	10436M c/s		
SHA1	2495M c/s	879M c/s	3833M c/s		
SHA256	1036M c/s	337M c/s	1413M c/s		
SHA512	179M c/s	103M c/s	383M c/s		
SHA-3(Keccak)	157M c/s	91M c/s	277M c/s		
RipeMD160	1690M c/s	615M c/s	2255M c/s		
Whirlpool	41281k c/s	94752k c/s	91128k c/s		
LM	1271M c/s	412M c/s	967M c/s		
NTLM	14789M c/s	4059M c/s	19011M c/s		
NetNTLMv1	7912M c/s	1844M c/s	8605M c/s		
NetNTLMv2	545M c/s	205M c/s	491M c/s	135M c/s	6456M c/s
WPA/WPA2	130k c/s	47k c/s	181k c/s	54k c/s	1454k c/s

For example, PC5 can do 174152M c/s against NTLM, that is 174152000000 tries per second.

[[John the Ripper benchmarks]]

Old revisions

Trace: • John the Ripper user community resources • John the Ripper benchmarks

John the Ripper benchmarks

Initially, this page will be the place to collect and share trivial `john -test` benchmarks on different systems. At a later tim sense to turn it into a namespace with sub-pages for `john -test` benchmarks (only c/s rate matters) and actual cracki things matter). Also, the underlying data may be uploaded/collected (e.g., exact `john -test` outputs, /proc/cpuinfo off of john.log files).

Please add your benchmark results to the tables below as appropriate. Please make sure to run the benchmarks on an **system**. For OpenMP- and MPI-enabled benchmarks, pick the "real" c/s rate. For single CPU core benchmarks, the "real" ar results should be almost the same (as long as the system is indeed otherwise idle), so it should not matter which one of pick.

Please keep these tables sorted by performance at DES-based crypt() for "many salts", better results listed first, best results for CPUs operating at their rated frequencies (non-overclocked) and running non-modified JtR code shown in ***b*** absolute best results).

Collected "john --test" benchmarks for OpenMP-enabled builds

DES crypt() many / one salt	MD5 crypt()	bcrypt x32	Windows LanMan	CPUs & clock rate	threads	logical CPUs/ physical cores	JtR	OS	compiler
117315K / 23130K 128/128 BS SSE2-16	**1953K** 128/128 SSE2 intrinsics 12x [1]	**56320** 32/64 X2	33410K [2] 128/128 BS SSE2-16	16x X7550 2.0 GHz HT disabled [3]	128	128 / 128	1.7.9-jumbo-6'ish bleeding-jumbo	Linux	gcc 4.7.0
73334K / 27426K 128/128 BS SSE2-16	850944 SSE2i 12x	23808 32/64 X2	42630K 128/128 BS SSE2-16	4x X7560 2.27 GHz	64	64 / 32	1.7.9-jumbo-5	Linux	gcc 4.4.6

Microsoft

Existing Data Points

- Limited to PCs and PC-like hardware
- Are these benchmarks on the attacker's hardware?

- Should there be an organized set of benchmarks/hardware?
  - Maybe something like *"eBACS: ECRYPT Benchmarking of Cryptographic Systems"*
  - What functions should be benchmarked?
  - With what parameters?
  - On which hardware?
  - How to incentivize contributions?

Microsoft

Explicitly formulated hypotheses

- For public key crypto, things like
  - Moore's law
  - Improvements in networking and memory (esp. for factoring)
  - Algorithmic advances

- For password hashing
  - What type(s) of hardware will the attacker use?
  - How will compute/memory resources change over time?
  - Algorithmic improvements in crackers
    - Implementation techniques
    - Techniques to generate candidate passwords

Microsoft

Formal Security Analysis

- Most PHC submissions do very little security analysis ☹
- Ideally, each would have a theorem something like

  If assumption X holds, then candidate Y is a foo.

  where *X* is well-defined assumption and *foo* is useful property for a PH function to have (secure KDF, memory hard)
- This type of analysis may be easier when *Y* is constructed using existing, well-studied primitives
- A security proof is not a hard requirement, but it can provide some assurance, and help understanding of the design
- How much cryptanalysis is being attempted on PHC candidates?
  - Hopefully this increases as the number of candidates decreases

Microsoft

Thank you!

- Summary
- Requirements
- Parameter selection
- Usability
- Q&A
- Engineering practicalities
- Security

Marsh Ray @marshray
maray@microsoft.com

Greg Zaverucha
gregz@microsoft.com

Microsoft