

13. Authentication

Part 1



Blase Ur and Grant Ho
February 19th, 2024
CMSC 23200



THE UNIVERSITY OF
CHICAGO

Who Am I?

- Grant Ho
 - Distinguished security researcher
 - Recently moved here from California; hates the cold
 - Fan of hot dogs
 - Ed course forum expert

Or Am I?

How (and why) do we
authenticate users?

Authentication in the Abstract

- **Principal:** legitimate owner of an identity
- **Claimant:** entity trying to be authenticated
- Verify that **people** or **things** (e.g., server) are who they claim, or maybe that the claimant has some **attribute**
- Authentication \neq Authorization \neq Access Control
 - *Authorization* is deciding whether an entity should have access to a given resource
 - *Access control* lists / policies

Authentication Use Cases

- Explicit authentication
 - Single-factor authentication
 - Multi-factor authentication (e.g., with Duo)
- Implicit authentication
 - Continuous authentication (e.g., with behavioral biometrics)
- Risk-based authentication: vary auth requirements based on estimated risk

How We Authenticate (1/2)

- Something you know
 - Password
 - PIN (Personal Identification Number)
- Something you have
 - Private key (of a public-private key pair)
 - Hardware device (often with a key/seed)
 - Phone (running particular software)
 - Token (e.g., string stored in a cookie)

How We Authenticate (2/2)

- Something you are
 - Biometrics (e.g., iris or fingerprint)
 - Behavioral tendencies (behavioral biometrics)
- Somewhere you are
 - Location-limited channels
 - IP address
- Someone you know (social authentication)
- Some system vouches for you
 - Single sign-on (e.g., UChicago shib/Okta)
 - PKI Certificate Authorities

[illegible]

Why Are Passwords So Prevalent?

- Easy to use
- Easy to deploy
- Nothing to carry
- No “silver-bullet” alternative

Why Are Passwords So Prevalent?

<i>Memorywise-Effortless</i>	Usability
<i>Scalable-for-Users</i>	
<i>Nothing-to-Carry</i>	
<i>Physically-Effortless</i>	
<i>Easy-to-Learn</i>	
<i>Efficient-to-Use</i>	
<i>Infrequent-Errors</i>	
<i>Easy-Recovery-from-Loss</i>	
<i>Accessible</i>	Deployability
<i>Negligible-Cost-per-User</i>	
<i>Server-Compatible</i>	
<i>Browser-Compatible</i>	
<i>Mature</i>	
<i>Non-Proprietary</i>	
<i>Resilient-to-Physical-Observation</i>	Security
<i>Resilient-to-Targeted-Impersonation</i>	
<i>Resilient-to-Throttled-Guessing</i>	
<i>Resilient-to-Unthrottled-Guessing</i>	
<i>Resilient-to-Internal-Observation</i>	
<i>Resilient-to-Leaks-from-Other-Verifiers</i>	
<i>Resilient-to-Phishing</i>	
<i>Resilient-to-Theft</i>	
<i>No-Trusted-Third-Party</i>	
<i>Requiring-Explicit-Consent</i>	
<i>Unlinkable</i>	

Bonneau et al. "The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes," In *Proc. IEEE S&P*, 2012

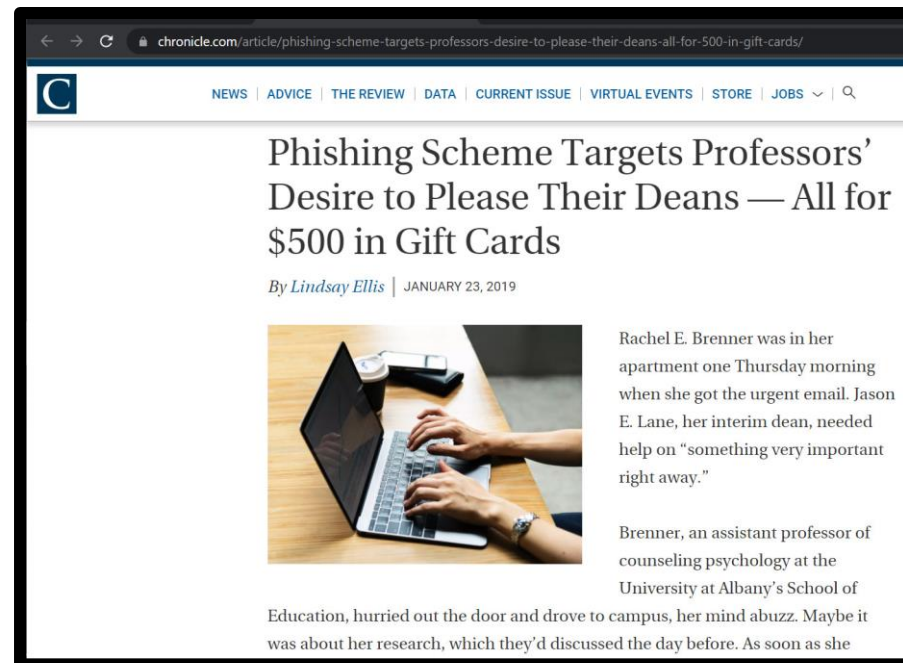
Why Are Passwords So Prevalent?

Category	Scheme	Described in section	Reference	Usability							Deployability					Security											
				Memorywise-Effortless	Scalable-for-Users	Nothing-to-Carry	Physically-Effortless	Easy-to-Learn	Efficient-to-Use	Infrequent-Errors	Easy-Recovery-from-Loss	Accessible	Negligible-Cost-per-User	Server-Compatible	Browser-Compatible	Mature	Non-Proprietary	Resilient-to-Physical-Observation	Resilient-to-Targeted-Impersonation	Resilient-to-Throttled-Guessing	Resilient-to-Unthrottled-Guessing	Resilient-to-Internal-Observation	Resilient-to-Leaks-from-Other-Verifiers	Resilient-to-Phishing	Resilient-to-Theft	No-Trusted-Third-Party	Requiring-Explicit-Consent
(Incumbent)	Web passwords	III	[13]		●			●	●	○	●	●	●	●	●	●	○						●	●	●	●	
Password managers	Firefox	IV-A	[22]	●	●	○	○	●	●	●	■	●	●	●	■	●	●	○	○				●	●	●	●	
	LastPass		[42]	●	●	○	○	●	●	●	○	●	●	●	■	●	●	○	○	○	○	○	●	●	■	●	
Proxy	URRSA	IV-B	[5]	●		■		●	■	○	■	■	●	●	○	■	■	○				○		●	■	■	●
	Impostor		[23]	●	●	●		●	■	■	■	■	●	●	○	■	■	○				○		●	●	■	■
Federated	OpenID	IV-C	[27]	●	●	●	○	○	●	●	■	●	●	■	●	●	●	○	○	○	○		●		●	■	■
	Microsoft Passport		[43]	●	●	●	○	●	●	●	■	●	■	■	●	●	■	○	○	○	○		●		●	■	■
	Facebook Connect		[44]	●	●	●	○	●	●	●	■	●	●	■	●	●	■	○	○	○	○		●		●	■	■
	BrowserID		[45]	●	●		○	●	●	●	■	●	●	■	○	○	■	○	○	○	○		●		●	■	■
	OTP over email		[46]	●	●	●		●	■	■	■	●	●	■	●	●	■	○	○	○	○		●	●	■	■	■
Graphical	PCCP	IV-D	[7]		●		●	○	○		■	●	■	●	■	●	●	●	○	○		●		●	●	●	
	PassGo		[47]		●		●	○	○		■	●	■	●	○	■	●	○	○				●		●	●	●
Cognitive	GrIDsure (original)	IV-E	[30]		●		●	○	○		■	■	■	■	■	■	■		●					●	●	●	●
	Weinshall		[48]		●		■	■	■	■	■	■	■	■	■	■	○	○				●		●	●	●	●
	Hopper Blum		[49]		●		■	■	■	■	■	■	■	■	■	■	○	○				●		●	●	●	●
	Word Association		[50]		●		●	○	○		■	■	■	■	■	■	■	■		■				●	●	●	●
Paper tokens	OTPW	IV-F	[33]		■		●	■	■		■	●	■	●	●	●	●	●	●	●	●	●	●	●	●	●	●
	S/KEY		[32]	●	■		●	■	○	●	■	●	■	●	●	●	●	●	●	●	○	■	●	●	●	●	●

Bonneau et al. "The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes," In *Proc. IEEE S&P*, 2012

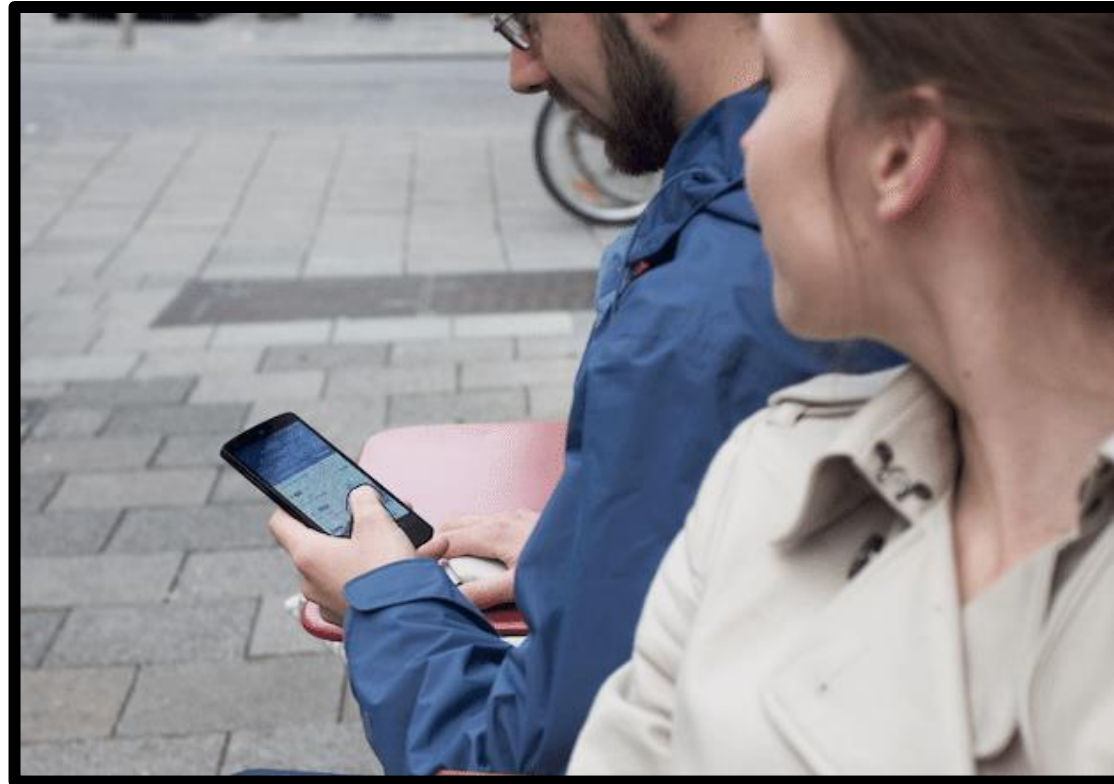
Attacks Against Passwords

- **Phishing** attack: try to trick the user into giving their credentials to you, believing you are the legitimate system
 - **Spear phishing**: targeted to the recipient



Attacks Against Passwords

- **Shoulder surfing:** looking at someone else entering their credentials



Detour: Storing Passwords

- Goal: Prevent attacker from being able to use a stolen password database immediately (without more work)
- Hash function: one-way function
 - Traditionally designed for efficiency (e.g., MD5, SHA-2), but don't ever use those!
 - Use password-specific hash functions (e.g., bcrypt, scrypt, Argon2)

Hashing on one NVIDIA RTX 4090

- Hashcat benchmarks
- MD5: ~ 150 billion / second
- SHA-1: ~ 50 billion / second
- UNIX md5crypt: ~ 60 million / second
- NTLM: ~ 250 billion / second
- SHA-2 (256): ~ 20 billion / second
- bcrypt (32 iterations): ~ 240,000 / second
- scrypt (16,384 iterations): ~ 7,000 / second

Storing Passwords

- **Salt**: random string assigned per-user
 - Combine the password with the salt, then hash it
 - Stored alongside the hashed password
 - Prevents the use of rainbow tables (hash outputs are precomputed for many passwords, mapping sorted by *output*)
 - Increases the attacker's work proportional to the number of accounts
- **Pepper**: secret salt (very uncommon)
- Both **salt** and **hash** passwords

Typical (Web) Account Creation

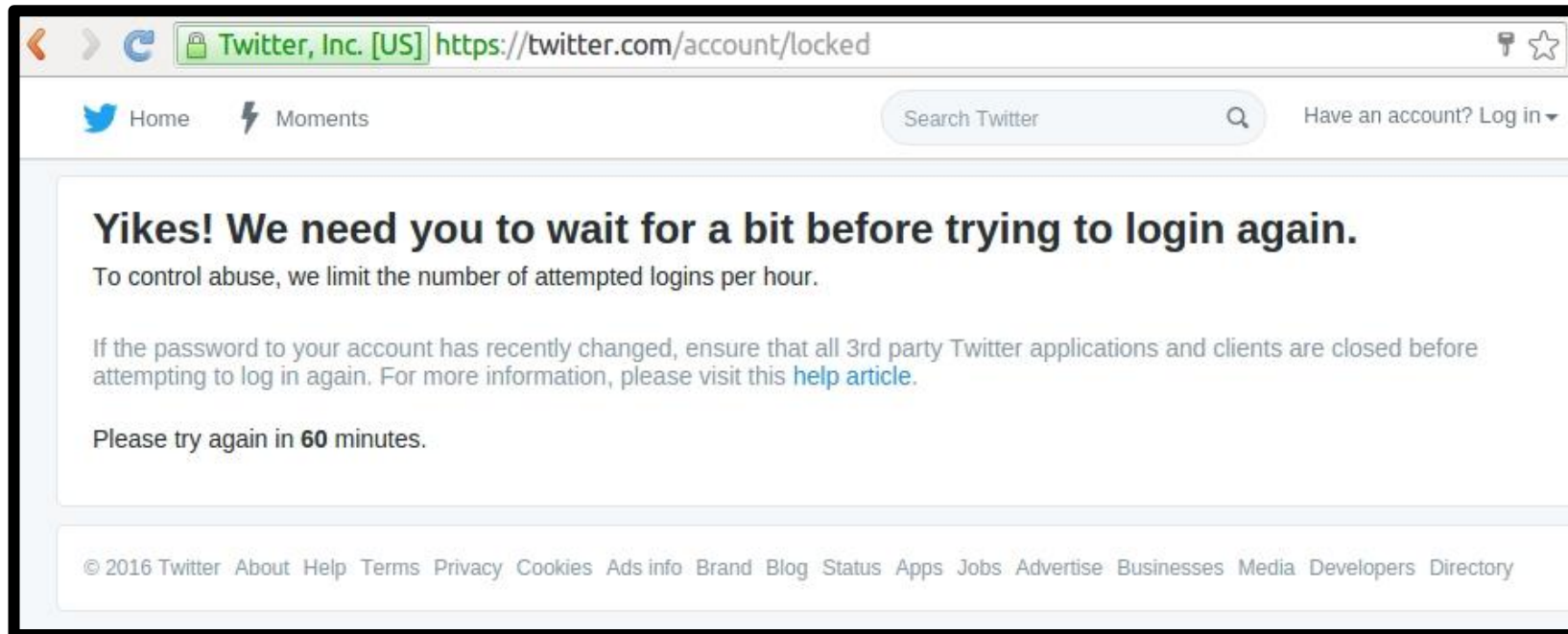
- User sends **username** and desired **password** over an encrypted tunnel
- Server validates username (e.g., does it exist in the system?) and password (e.g., does it meet composition requirements?)
- Server generates a random salt
 - Think about how long the salt should be!
- Server stores **username**, **salt**, and **hash(password|salt)** in database

Typical (Web) Authentication

- User sends **username** and **password₀** over an encrypted tunnel
- Server looks up the salt and hash output associated with that username
- Server computes $\text{hash}(\text{password}_0 | \text{salt})$
- If it matches the hash output in the database, typically send back auth token (long string attacker can't guess associated with that user's session)

Guessing Attacks Against Passwords

- **Online attack (web)**
 - Try passwords on a live system
 - Usually rate-limited



Guessing Attacks Against Passwords

- **Online attack** (web)
 - Try passwords on a live system
 - Usually rate-limited
- Authenticating to a device is often similarly rate-limited (e.g., iPhone PIN) using secure hardware

Guessing Attacks Against Passwords

- **Offline attack** (web)
 - Try to guess passwords from a stolen copy of the password store or password database

Guessing Attacks Against Passwords

- **Offline attack** (web)
 - Try to guess passwords from a stolen copy of the password store or password database
- Attacking a file encrypted using a key derived from a password (e.g., with PBKDF2) is similar

Offline Attack (In Practice)

- Attacker compromises database (e.g., via SQL injection)
 - hash("Blase") =

`$2a$04$iHdEgkI681VdDMc3f7edau9phRwORvhYjqWAIb7hb4B5uFJO1g4zi`

\$ = delimiter

2a = bcrypt

04 = 2^4 iterations (cost)

iHdEgkI681VdDMc3f7edau = 16 bytes of salt (radix-64 encoded)

9phRwORvhYjqWAIb7hb4B5uFJO1g4zi = 24 bytes of hash output (radix-64 encoded)

- Attacker makes guesses (from most likely/probable to the least) and hashes those guesses
- Finds match → try on other sites
 - Password **reuse** is a core problem

Attack Model



80d561388725fa74f2d03cd16e1d687c



Attack Model



80d561388725fa74f2d03cd16e1d687c



1. $h("123456") = e10adc3949ba59abbe56e057f20f883e$

Attack Model



80d561388725fa74f2d03cd16e1d687c



1. $h("123456") = e10adc3949ba59abbe56e057f20f883e$
2. $h("password") = 5f4dcc3b5aa765d61d8327deb882cf99$

Attack Model



80d561388725fa74f2d03cd16e1d687c



1. $h("123456") = e10adc3949ba59abbe56e057f20f883e$
2. $h("password") = 5f4dcc3b5aa765d61d8327deb882cf99$
3. $h("monkey") = d0763edaa9d9bd2a9516280e9044d885$

Attack Model



80d561388725fa74f2d03cd16e1d687c



1. $h("123456") = e10adc3949ba59abbe56e057f20f883e$
2. $h("password") = 5f4dcc3b5aa765d61d8327deb882cf99$
3. $h("monkey") = d0763edaa9d9bd2a9516280e9044d885$
4. $h("letmein") = 0d107d09f5bbe40cade3de5c71e9e9b7$

Attack Model



80d561388725fa74f2d03cd16e1d687c



1. $h("123456") = e10adc3949ba59abbe56e057f20f883e$
2. $h("password") = 5f4dcc3b5aa765d61d8327deb882cf99$
3. $h("monkey") = d0763edaa9d9bd2a9516280e9044d885$
4. $h("letmein") = 0d107d09f5bbe40cade3de5c71e9e9b7$
5. $h("p@ssw0rd") = 0f359740bd1cda994f8b55330c86d845$

Attack Model



80d561388725fa74f2d03cd16e1d687c



1. $h("123456") = e10adc3949ba59abbe56e057f20f883e$
2. $h("password") = 5f4dcc3b5aa765d61d8327deb882cf99$
3. $h("monkey") = d0763edaa9d9bd2a9516280e9044d885$
4. $h("letmein") = 0d107d09f5bbe40cade3de5c71e9e9b7$
5. $h("p@ssw0rd") = 0f359740bd1cda994f8b55330c86d845$
6. $h("Chic4go") = \mathbf{80d561388725fa74f2d03cd16e1d687c}$






Credential Breaches

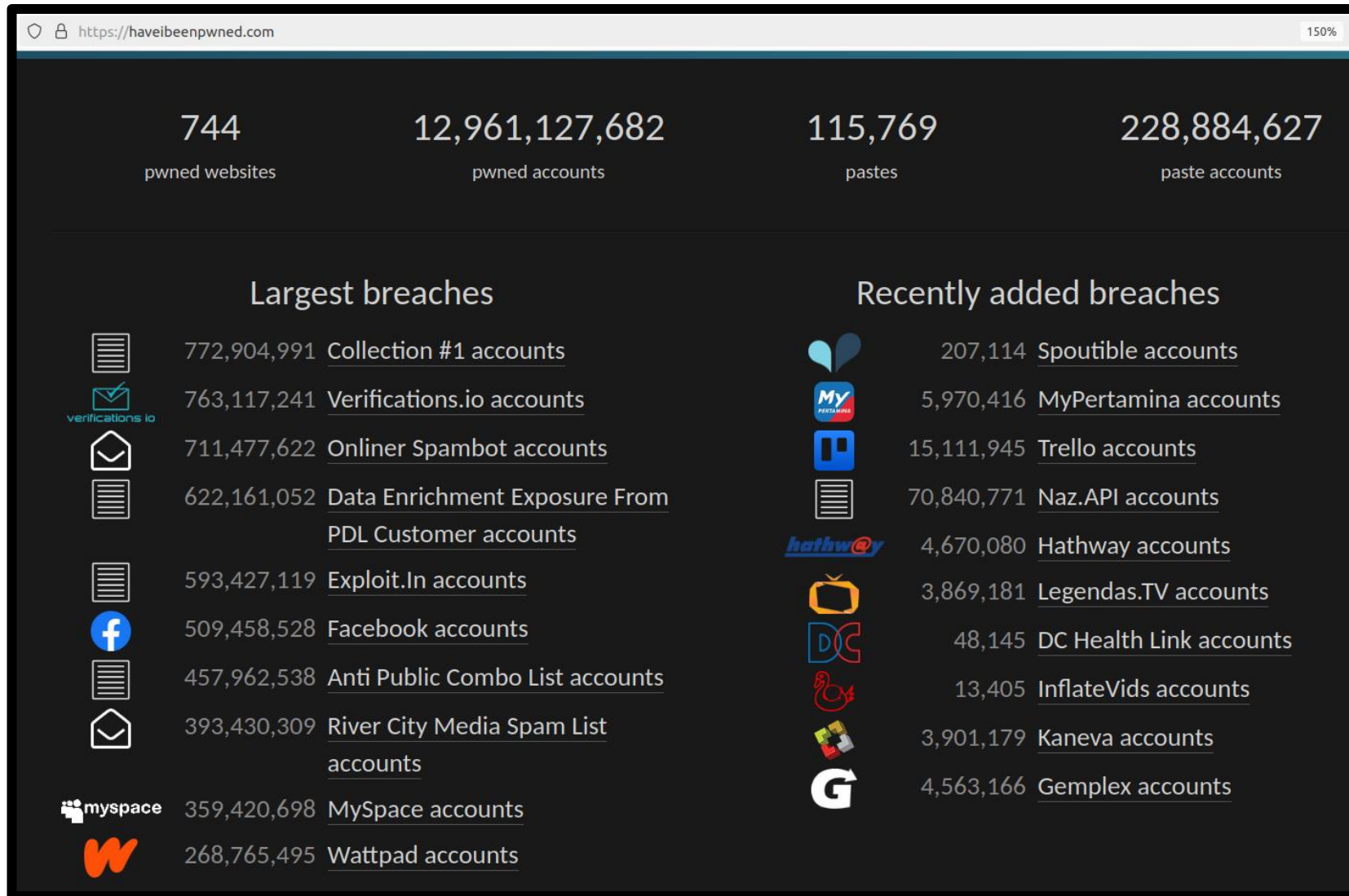
Some Breached Companies



Data-Driven Statistical Attacks

- (2009) 32 million passwords: 
- (2016) 117 million passwords: 
- (2017) 3 billion passwords: 
 - Still not released publicly as of 2024

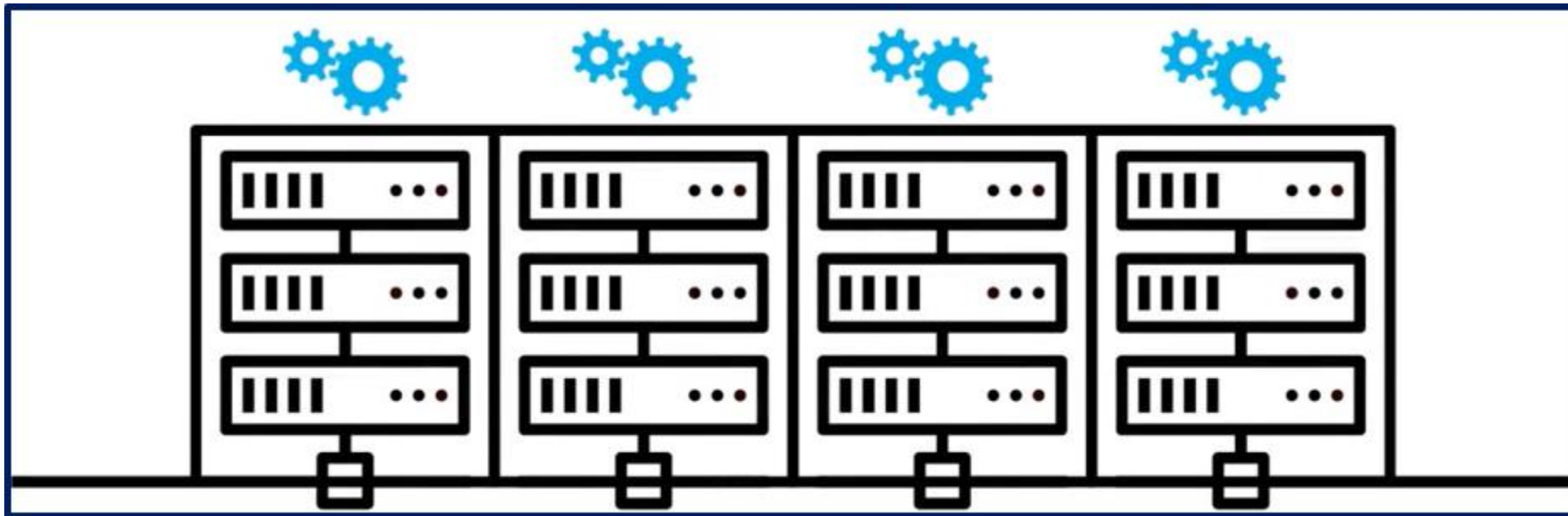
Have I Been Pwned (as of 2/19/24)



Password Policies (Partial Attempt to Combat Attacks)

What Do Human-Chosen Passwords Look Like? (Live Demo)

Password Cracking



Blase Ur, Sean M. Segreti, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, Saranga Komanduri, Darya Kurilova, Michelle L. Mazurek, William Melicher, Richard Shay. Measuring Real-World Accuracies and Biases in Modeling Password Guessability. In *Proc. USENIX Security Symposium*, 2015.

Statistical Metrics For Passwords

- Traditionally: Shannon entropy
- Recently: α -guesswork
- Disadvantages of statistical approaches
 - Entropy does not consider human tendencies
 - Usually no per-password estimates
 - Huge sample required for accuracy (since many passwords are related to each other)
 - Does not model real-world attacks

Parameterized Guessability

- How many guesses a particular cracking algorithm with particular training data would take to guess a password

Parameterized Guessability

Chic4go

Guess # 6

Parameterized Guessability

j@mesb0nd007!

Guess # 366,163,847,194

Parameterized Guessability

$n (c \$ J Z X ! z K c ^ b I A X ^ N$

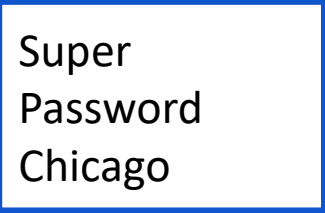
Guess # past cutoff

Some Key Password-Cracking Approaches

- Brute force (or selective brute force)
- Wordlist
- Mangled wordlist
 - Hashcat and John the Ripper
- Markov models
- Probabilistic Context-Free Grammar
- Deep learning
- In practice: manual, iterative updates

Mangled Wordlist Attack

Wordlist



Super
Password
Chicago

Mangled Wordlist Attack

Wordlist

Super
Password
Chicago

Rulelist

1. Append "1"
2. Replace "a" → "4"
3. Lowercase all

Mangled Wordlist Attack

Wordlist

Super
Password
Chicago

Rulelist

1. Append "1"
2. Replace "a" → "4"
3. Lowercase all

Guesses

Super1

Mangled Wordlist Attack

Wordlist

Super
Password
Chicago

Rulelist

1. Append "1"
2. Replace "a" → "4"
3. Lowercase all

Guesses

Super1
Password1

Mangled Wordlist Attack

Wordlist

Super
Password
Chicago

Rulelist

1. Append "1"
2. Replace "a" → "4"
3. Lowercase all

Guesses

Super1
Password1
Chicago1

Mangled Wordlist Attack

Wordlist

Super
Password
Chicago

Rulelist

1. Append "1"
2. Replace "a" → "4"
3. Lowercase all

Guesses

Super1
Password1
Chicago1
Super
P4ssword
Chic4go

Mangled Wordlist Attack

Wordlist

Super
Password
Chicago

Rulelist

1. Append "1"
2. Replace "a" → "4"
3. Lowercase all

Guesses

Super1
Password1
Chicago1
Super
P4ssword
Chic4go
super
password
chicago

Example Wordlists and Rulelists

Wordlist

PGS ($\approx 20,000,000$)

Linkedin ($\approx 60,000,000$)

HIBP ($\approx 500,000,000$)

Example Wordlists and Rulelists

Wordlist

PGS ($\approx 20,000,000$)

Linkedin ($\approx 60,000,000$)

HIBP ($\approx 500,000,000$)

Rulelist

Korelogic ($\approx 5,000$)

Megatron ($\approx 15,000$)

Generated2 ($\approx 65,000$)

Example Wordlists and Rulelists

Wordlist

PGS ($\approx 20,000,000$)

Linkedin ($\approx 60,000,000$)

HIBP ($\approx 500,000,000$)

Rulelist

Korelogic ($\approx 5,000$)

Megatron ($\approx 15,000$)

Generated2 ($\approx 65,000$)



$10^9 - 10^{15}$
guesses

Example Wordlists and Rulelists

Wordlist

PGS ($\approx 20,000,000$)

Linkedin ($\approx 60,000,000$)

HIBP ($\approx 500,000,000$)

Rulelist

Korelogic ($\approx 5,000$)

Megatron ($\approx 15,000$)

Generated2 ($\approx 65,000$)



$10^9 - 10^{15}$
guesses

+ Hackers' private word/rule lists

John the Ripper

- Wordlist mode requires:
 - Wordlist (passwords and dictionary entries)
 - Mangling rules
- Guesses variants of input wordlist
- Speed: Fast
- “JTR”



John the Ripper



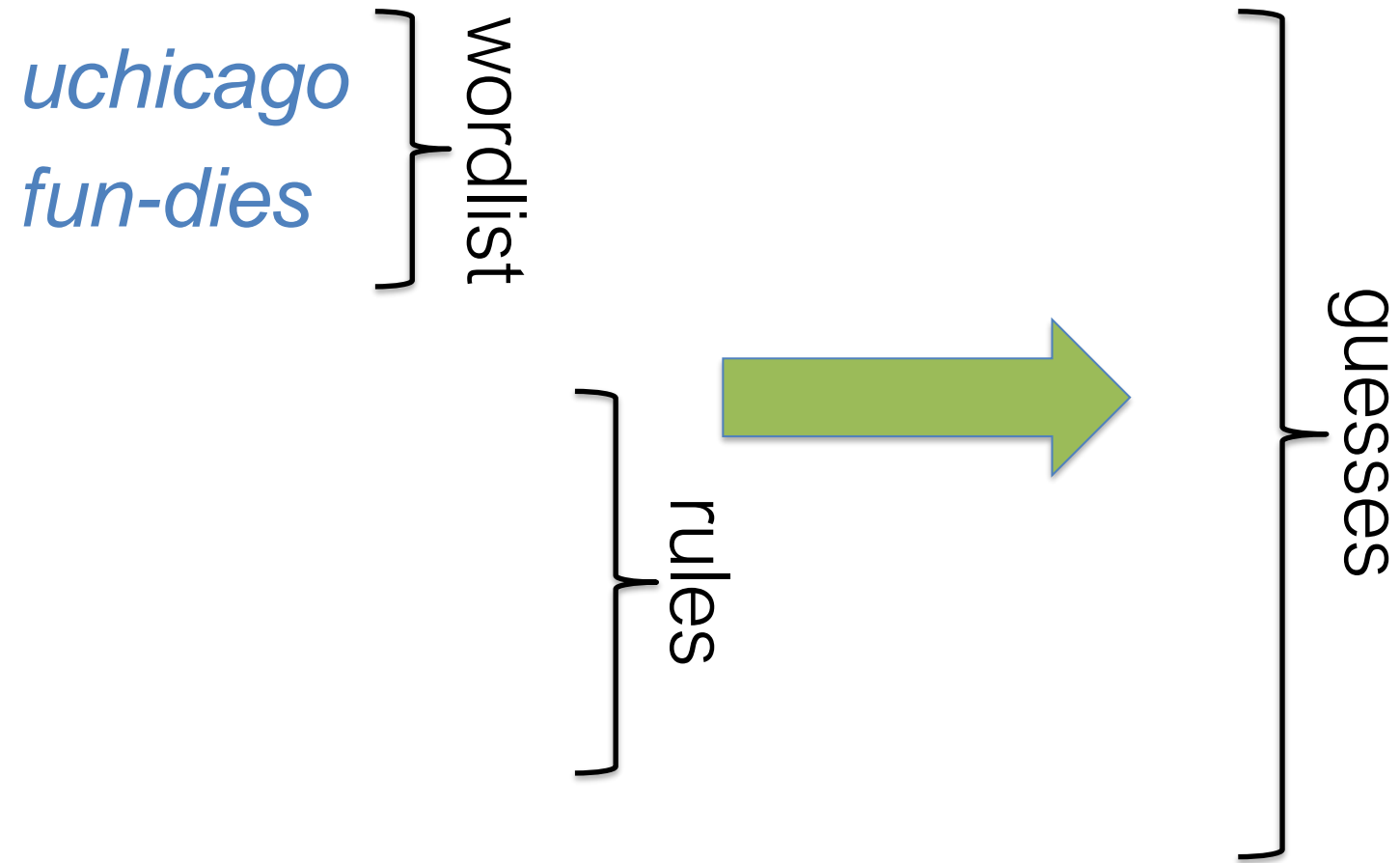
wordlist

rules

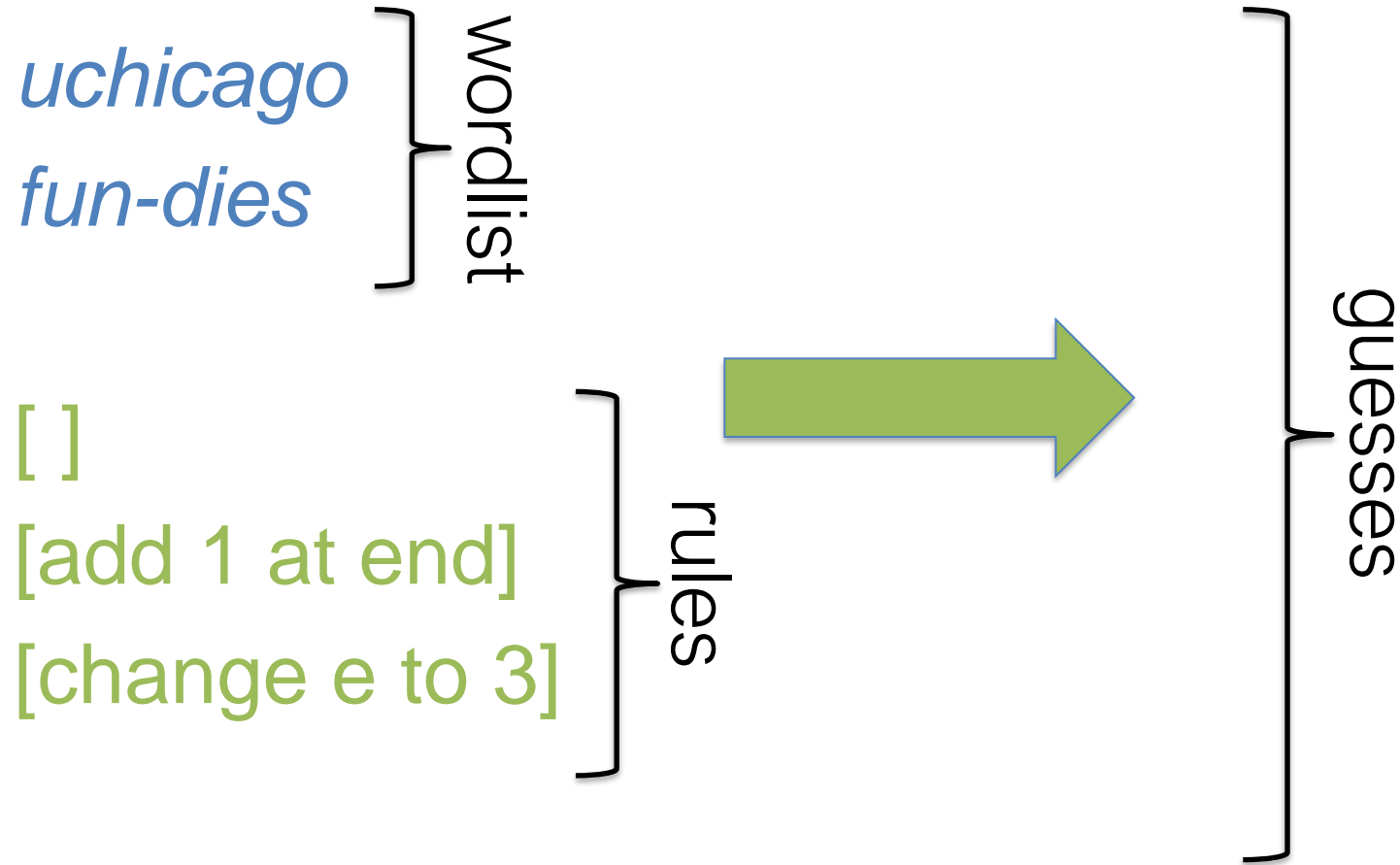


guesses

John the Ripper



John the Ripper



John the Ripper



uchicago
fun-dies

wordlist

[]

[add 1 at end]

[change e to 3]

rules

uchicago
fun-dies

guesses

John the Ripper



uchicago
fun-dies

wordlist

[]

[add 1 at end]

[change e to 3]

rules

uchicago
fun-dies

uchicago1

fun-dies1

guesses

John the Ripper



uchicago
fun-dies

} wordlist

[]
[add 1 at end]
[change e to 3]

} rules

uchicago
fun-dies
uchicago1
fun-dies1
uchicago
fun-di3s

} guesses

Hashcat

- Wordlist mode requires:
 - Wordlist (passwords and dictionary entries)
 - Mangling rules
- Guesses variants of input wordlist
- (Many other modes)
- Speed: Fast



hashcat
advanced
password
recovery

Hashcat



hashcat
advanced
password
recovery

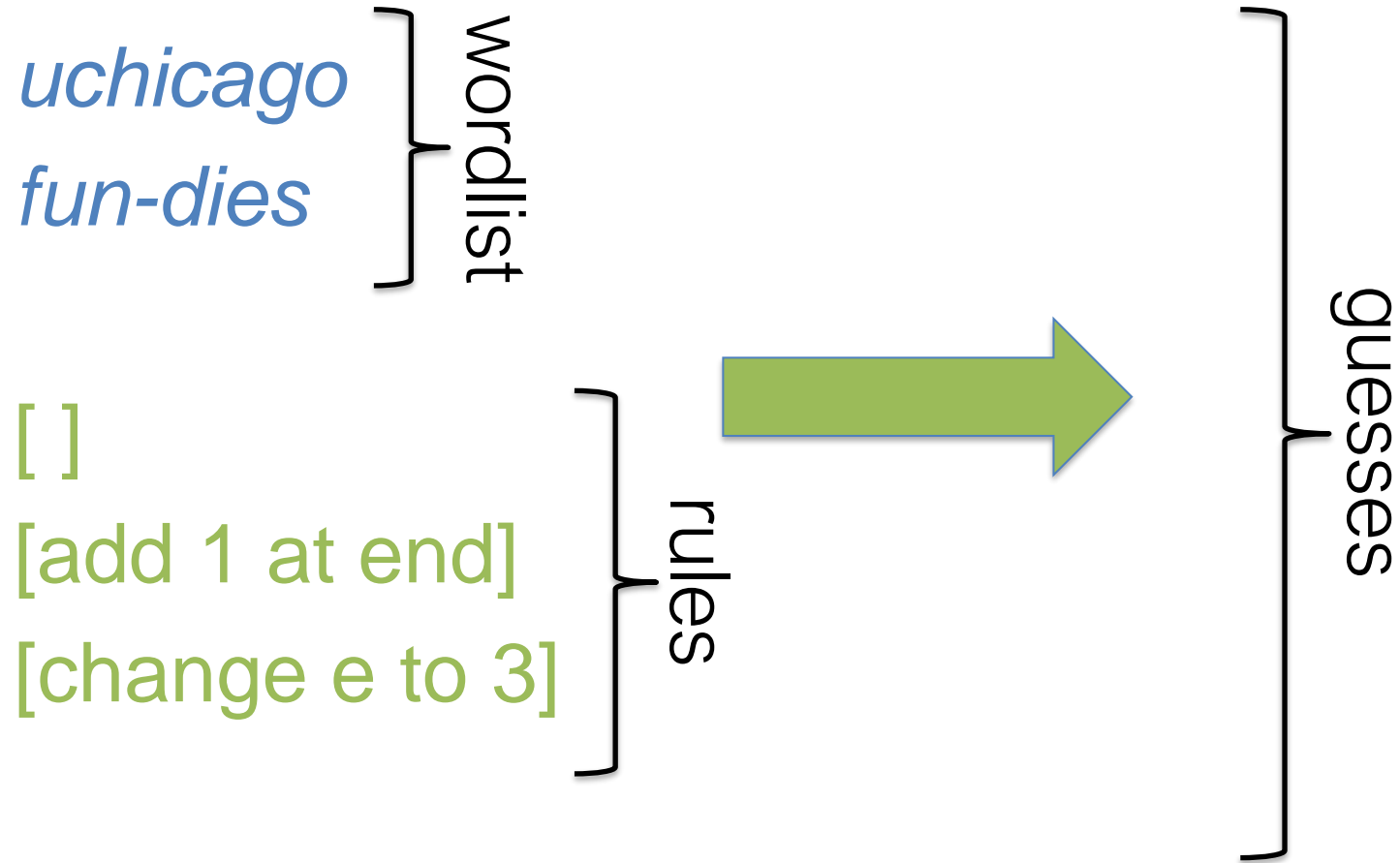
wordlist

rules



guesses

Hashcat



Hashcat



hashcat
advanced
password
recovery

uchicago
fun-dies

wordlist

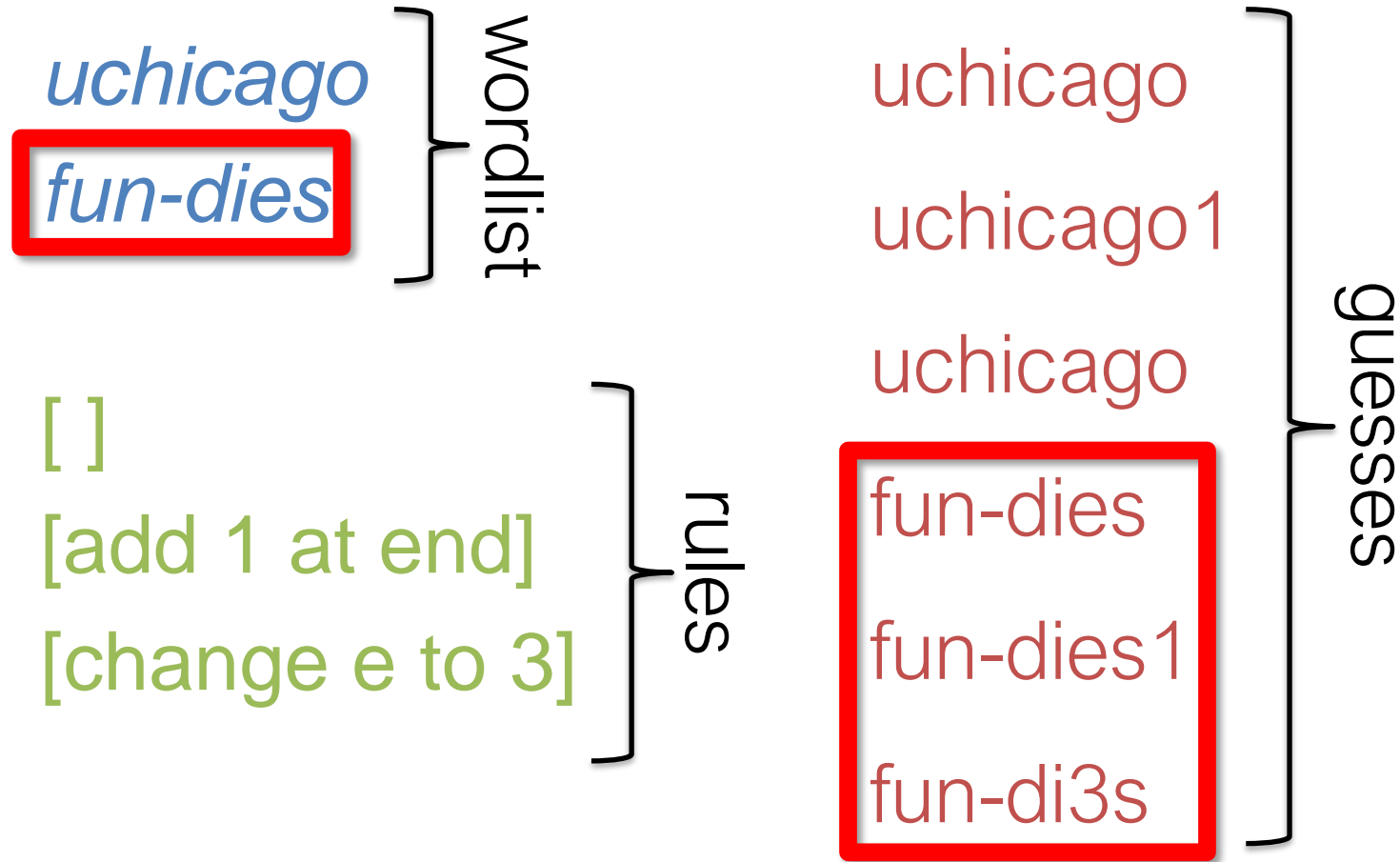
[]
[add 1 at end]
[change e to 3]

rules

uchicago
uchicago1
uchicago

guesses

Hashcat



Hashcat Mangling-Rule Language

Name	Function	Description	Example Rule	Input Word	Output Word	Note
Nothing	:	do nothing	:	p@ss-W0rd	p@ssW0rd	
Lower-case	l	Lowercase all letters	l	p@ss-W0rd	p@ssw0rd	
Upper-case	u	Uppercase all letters	u	p@ss-W0rd	P@SSWORD	
Capitalize	c	Capitalize the first letter and lower the rest	c	p@ss-W0rd	P@ssw0rd	
Invert Capitalize	C	Lowercase first found character, uppercase the rest	C	p@ss-W0rd	p@SSWORD	
Toggle Case	t	Toggle the case of all characters in word.	t	p@ss-W0rd	P@ssW0RD	
Toggle @	TN	Toggle the case of characters at position N	T3	p@ss-W0rd	p@ssW0rd	*
Reverse	r	Reverse the entire word	r	p@ss-W0rd	dr0Wss@p	
Duplicate	d	Duplicate entire word	d	p@ss-W0rd	p@ssW0rdp@ssW0rd	
Duplicate N	pN	Append duplicated word N times	p2	p@ss-W0rd	p@ssW0rdp@ssW0rdp@ssW0rd	
Reflect	f	Duplicate word reversed	f	p@ss-W0rd	p@ssW0rd-dr0Wss@p	
Rotate Left	{	Rotates the word left.	{	p@ss-W0rd	@ssW0rdp	
Rotate Right	}	Rotates the word right	}	p@ss-W0rd	dp@ssW0r	
Append Character	\$X	Append character X to end	\$1	p@ss-W0rd	p@ssW0rd1	
Prepend Character	^X	Prepend character X to front	^1	p@ss-W0rd	1p@ssW0rd	
Truncate left	[Deletes first character	[p@ss-W0rd	@ssW0rd	
Truncate right]	Deletes last character]	p@ss-W0rd	p@assW0r	
Delete @ N	DN	Deletes character at position N	D3	p@ss-W0rd	p@ssW0rd	*
Extract range	xNM	Extracts M characters, starting at position N	x04	p@ss-W0rd	p@ss	* #
Omit range	ONM	Deletes M characters, starting at position N	O12	p@ss-W0rd	psW0rd	*
Insert @ N	iNX	Inserts character X at position N	i4!	p@ss-W0rd	p@ssiW0rd	*
Over-write N	oNX	Overwrites character at position N with X	o3\$	p@ss-W0rd	p@ss\$W0rd	*
Truncate @ N	'N	Truncate word at position N	'6	p@ss-W0rd	p@ssW0	*
Replace	sXY	Replace all instances of X with Y	ss\$	p@ss-W0rd	p@\$SW0rd	
Purge	@X	Purge all instances of X	@s	p@ss-W0rd	p@W0rd	+

Name	Function	Description	Example Rule	Note
Reject less	<N	Reject plains if their length is greater than N	<G	*
Reject greater	>N	Reject plains if their length is less or equal to N	>8	*
Reject equal	_N	Reject plains of length not equal to N	_7	*
Reject contain	!X	Reject plains which contain char X	!z	
Reject not contain	/X	Reject plains which do not contain char X	/e	
Reject equal first	(X	Reject plains which do not start with X	(h	
Reject equal last)X	Reject plains which do not end with X)t	
Reject equal at	=NX	Reject plains which do not have char X at position N	=1a	*
Reject contains	%NX	Reject plains which contain char X less than N times	%2a	*
Reject contains	Q	Reject plains where the memory saved matches current word	rMrQ	e.g. for palindrome

Name	Function	Description	Example Rule	Input Word	Output Word	Note
Swap front	k	Swaps first two characters	k	p@ssW0rd	@pssW0rd	
Swap back	K	Swaps last two characters	K	p@ssW0rd	p@ssW0dr	
Swap @ N	*NM	Swaps character at position N with character at position M	*34	p@ssW0rd	p@sWs0rd	*
Bitwise shift left	LN	Bitwise shift left character @ N	L2	p@ssW0rd	p@æssW0rd	*
Bitwise shift right	RN	Bitwise shift right character @ N	R2	p@ssW0rd	p@9ssW0rd	*
Ascii increment	+N	Increment character @ N by 1 ascii value	+2	p@ssW0rd	p@tsW0rd	*
Ascii decrement	-N	Decrement character @ N by 1 ascii value	-1	p@ssW0rd	p?ssW0rd	*
Replace N + 1	.N	Replaces character @ N with value at @ N plus 1	.1	p@ssW0rd	psssW0rd	*
Replace N - 1	,N	Replaces character @ N with value at @ N minus 1	,1	p@ssW0rd	ppssW0rd	*
Duplicate block front	yN	Duplicates first N characters	y2	p@ssW0rd	p@p@ss-W0rd	*
Duplicate block back	YN	Duplicates last N characters	Y2	p@ssW0rd	p@ssW0r-drd	*
Title	E	Lower case the whole line, then upper case the first letter and every letter after a space	E	p@ssW0rdw0rld	P@ssw0rdW0rld	+
Title w/separator	eX	Lower case the whole line, then upper case the first letter and every letter after a custom separator character	e-	p@ssW0rd-w0rld	P@ssw0rd-W0rld	+

Hashcat Mangling-Rule Language

`*05 003 d '7`

Switch the first and the sixth char;

Delete the first three chars;

Duplicate the whole word;

Truncate the word to length 7;

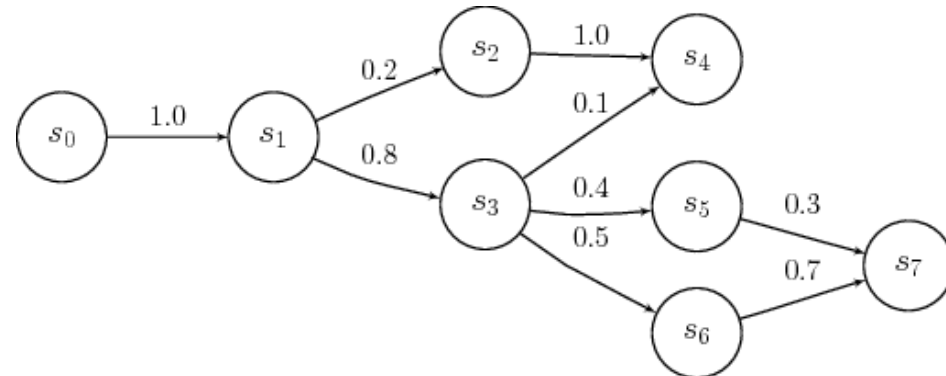
Hashcat (Other Modes)

- Mask attack (brute force within a specified character-class structure)
- Combinator attacks
- Hybrid attacks
- Many more!

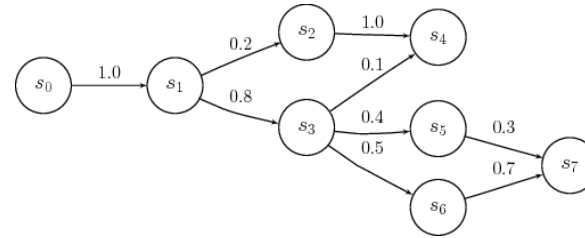


Markov Models

- Predicts future characters from previous
- Approach requires weighted data:
 - Passwords
 - Dictionaries
- Speed: Slow
- Smoothing is critical



Markov Models



chic4gooo

2-gram model (1 character of context):

[start] → c (1.0)

4 → g (1.0)

c → h (0.5), 4 (0.5)

g → o (1.0)

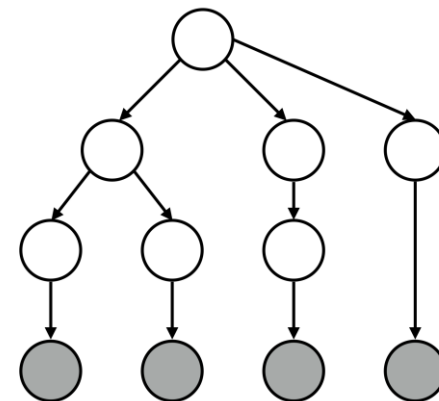
h → i (1.0)

i → c (1.0)

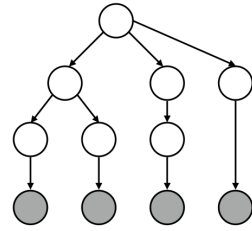
o → o (0.67) [end] (0.33)

Probabilistic Context-Free Grammar

- Generate password grammar
 - Structures
 - Terminals
- OG: Weir et al. IEEE S&P 2009
- Speed: Slow
- “PCFG”



PCFG



passwordpassword

password123

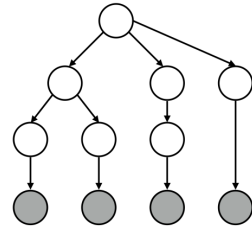
usenix3

5ecurity

iloveyou

nirvanaa123

PCFG



passwordpassword

*password*123

*unix*3

5*ecurity*

iloveyou

*nirvanaa*123

Structure Model:

L_{16} (1/6)

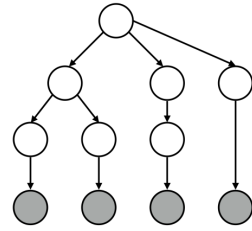
$L_8 D_3$ (2/6)

$L_6 D_1$ (1/6)

$D_1 L_7$ (1/6)

L_8 (1/6)

PCFG



passwordpassword

*password*123

*unix*3

5*ecurity*

iloveyou

*nirvanaa*123

Digit Model:

$D_1 \rightarrow 3 (0.5) 5 (0.5)$

$D_3 \rightarrow 123 (1.0)$

Repeat for letters, etc.

Professionals (“Pros”)

- Proprietary wordlists and configurations
 - 10^{14} guesses
 - Manually tuned, updated
- For example: KoreLogic
 - Password audits for Fortune 500 companies
 - Run DEF CON “Crack Me If You Can”

KoreLogic
S E C U R I T Y



Research Approach (2015)

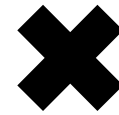
4 password sets

```
password  
iloveyou  
team0123  
...
```

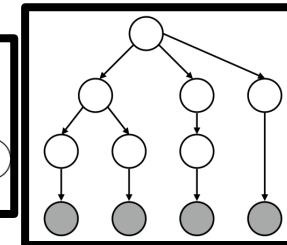
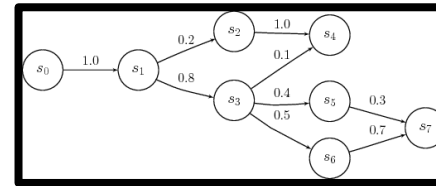
```
passwordpassword  
1234567812345678  
!1@2#3$4%5^6&7*8  
...
```

```
Pa$$w0rd  
iLov3you!  
1QaZ2W@x  
...
```

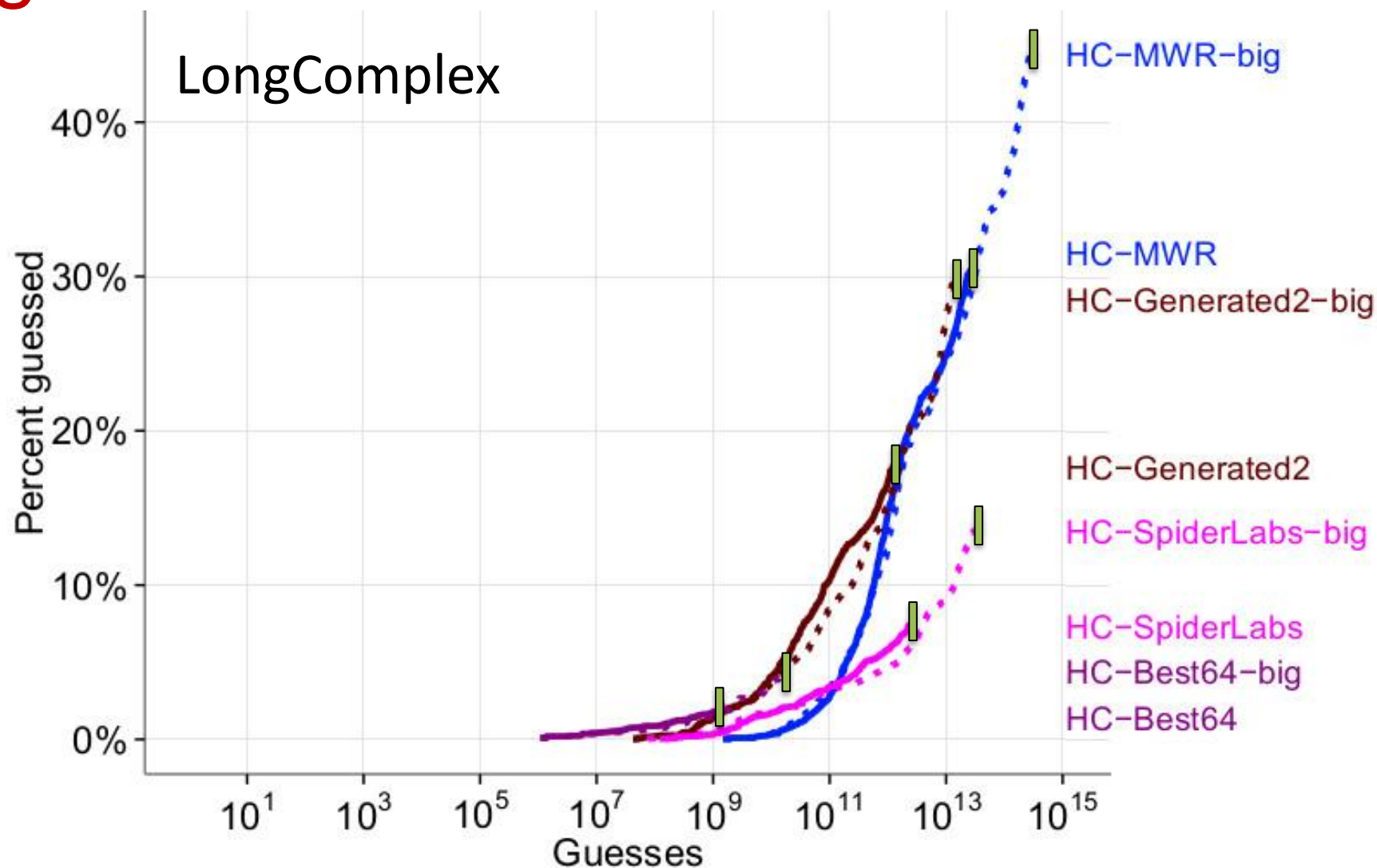
```
pa$$word1234  
12345678asDF  
!q1q!q1q!q1q  
...
```



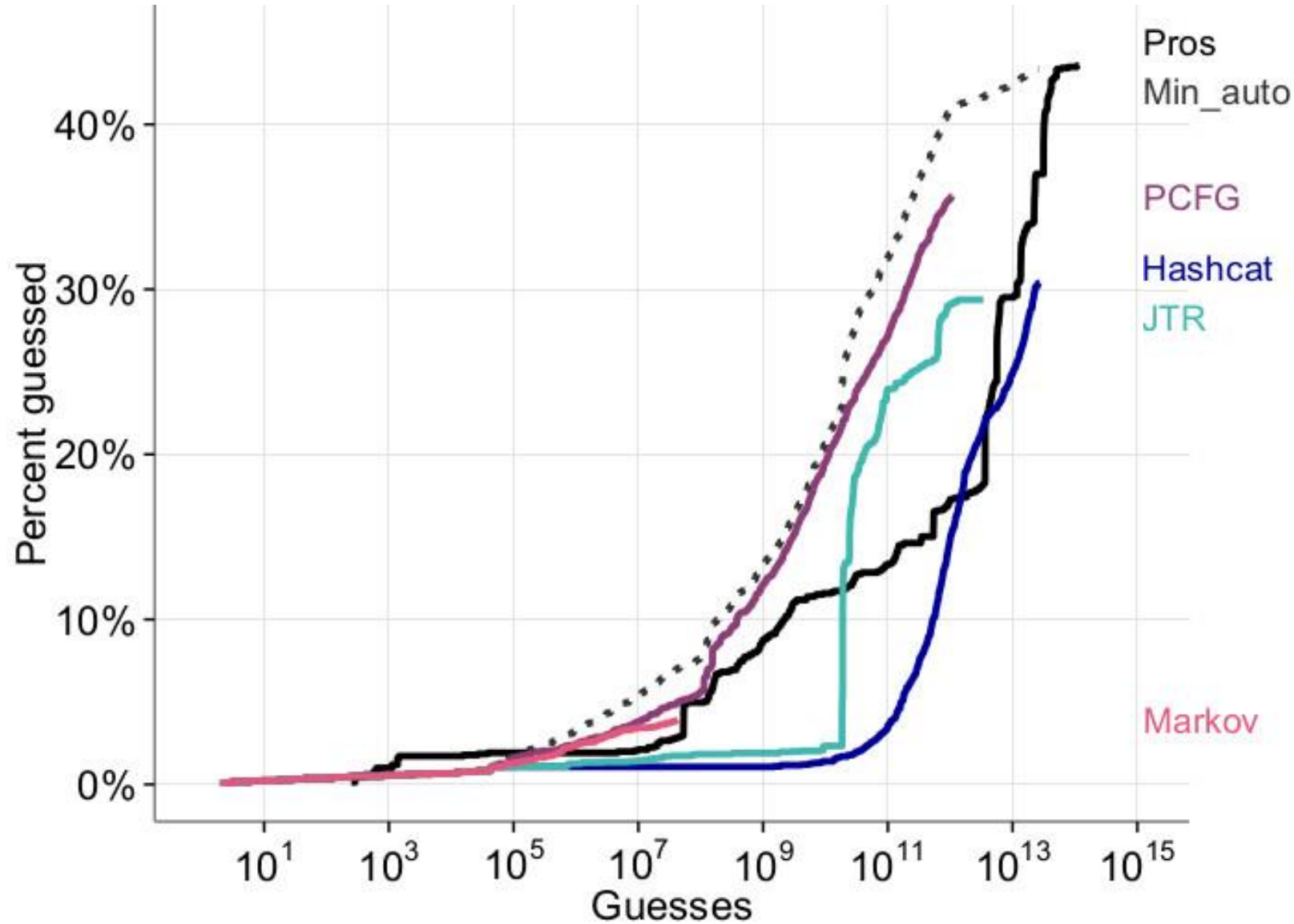
5 approaches



Configuration Is Crucial



Comparison for Complex Passwords



Per-Password Highly Impacted

P@ssw0rd!

Per-Password Highly Impacted

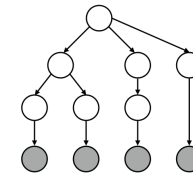
- JTR guess # 801



P@ssw0rd!

Per-Password Highly Impacted

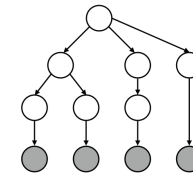
- JTR guess # 801
- Not guessed in 10^{14} PCFG guesses



P@ssw0rd!

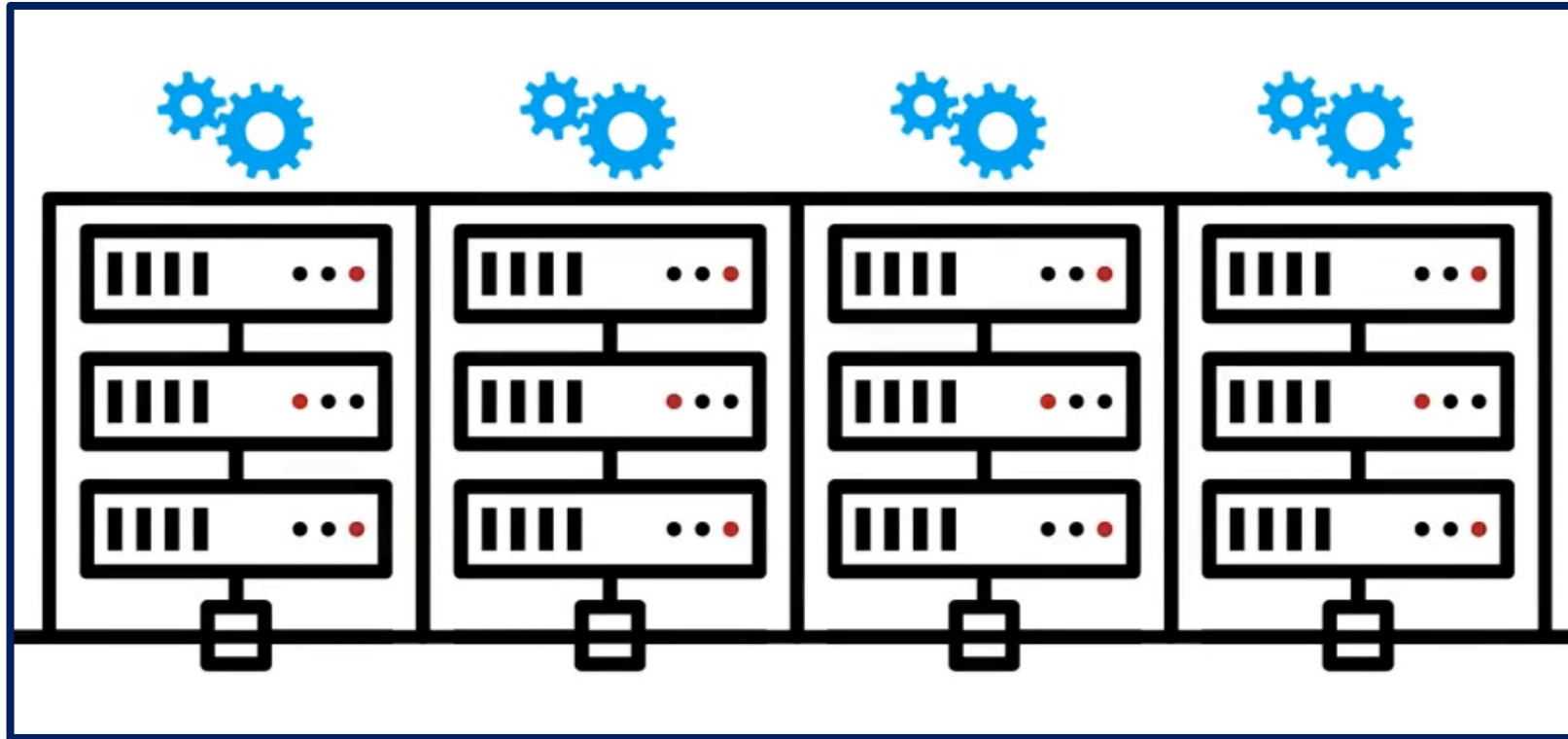
Per-Password Highly Impacted

- JTR guess # 801
- Not guessed in 10^{14} PCFG guesses



P@ssw0rd!

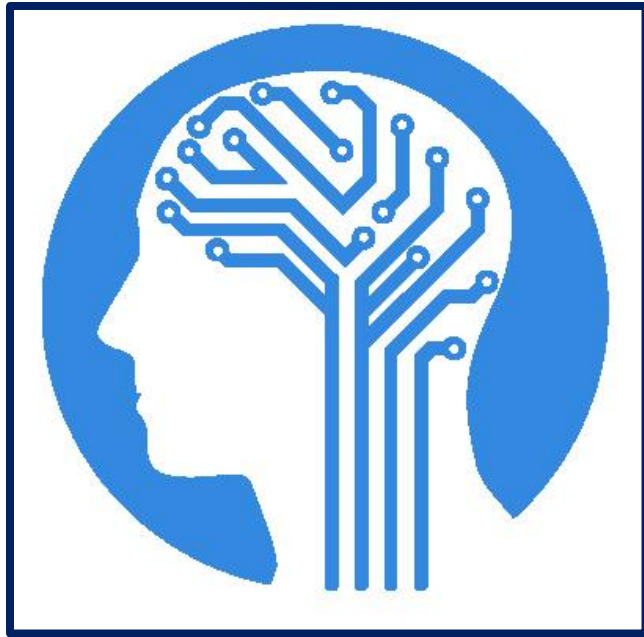
Neural Networks For Passwords



William Melicher, Blase Ur, Sean M. Segreti, Saranga Komanduri, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor. Fast, Lean, and Accurate: Modeling Password Guessability Using Neural Networks. In *Proc. USENIX Security Symposium*, 2016.

Better Password Scoring

- Real-time feedback
- Runs entirely client-side
- Accurately models password guessability



**Recurrent Neural
Networks (RNNs)**

LSTM Architecture

Generating Passwords

Generating Passwords

`passwd`  o or maybe 0 or O or ...

Generating Passwords

passw



Next char is:

A: 3%

B: 1%

C: 0.6%

...

O: 55%

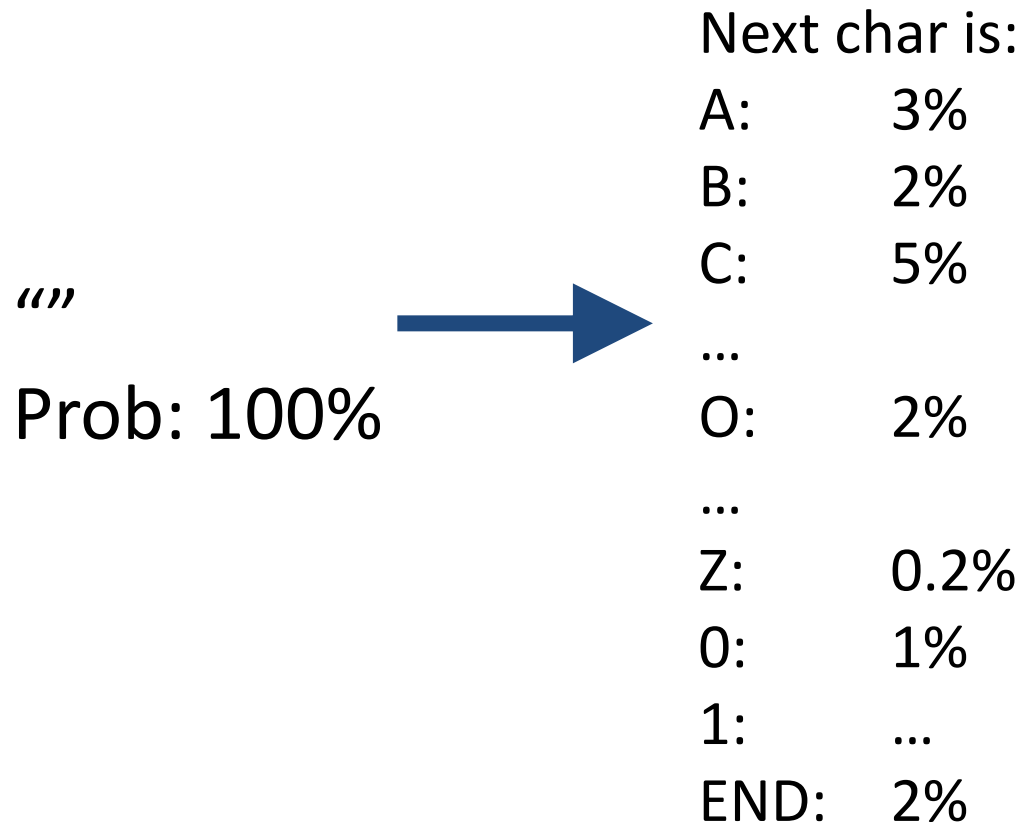
...

Z: 0.01%

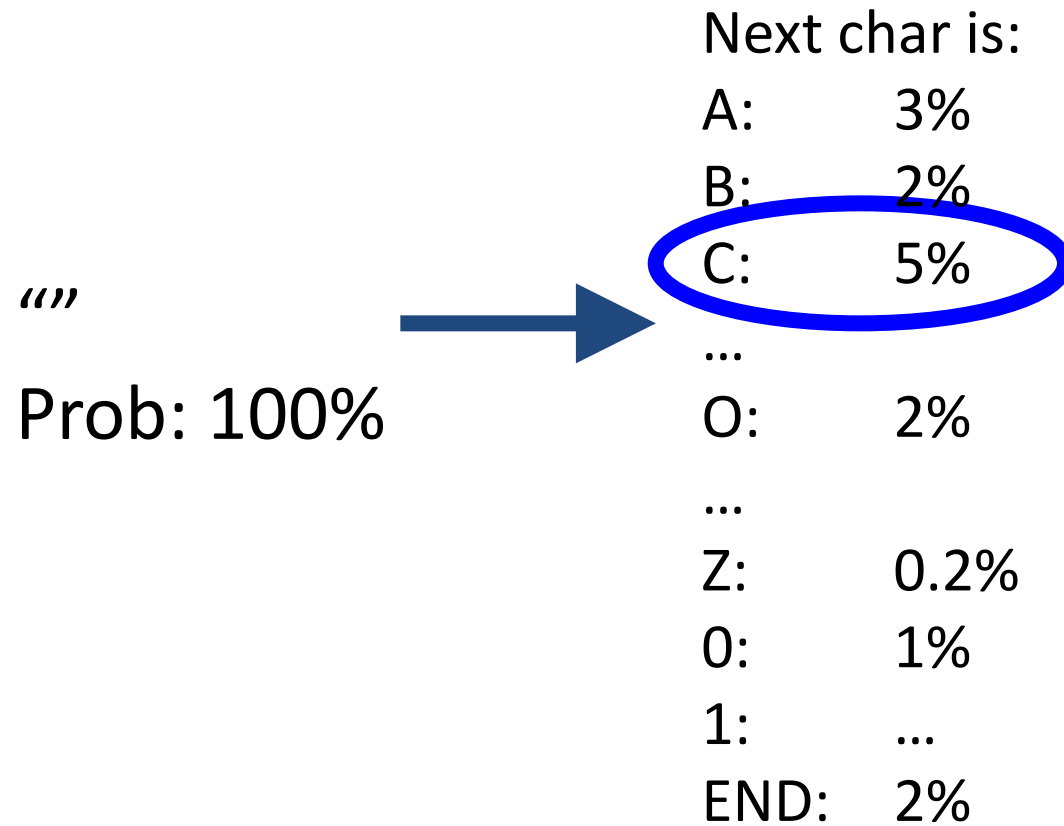
0: 20%

1: ...

Generating Passwords



Generating Passwords



Generating Passwords

“C”

Prob: 5%



Generating Passwords

“C”

Prob: 5%



Next char is:

A: 10%

B: 1%

C: 4%

...

O: 8%

...

Z: 0.02%

0: 3%

1: ...

END: 6%

Generating Passwords

“C”

Prob: 5%



Next char is:

A: 10%

B: 1%

C: 4%

...

O: 8%

...

Z: 0.02%

0: 3%

1: ...

END: 6%

Generating Passwords

“CA”

Prob: 0.5%



Next char is:

A: 3%

B: 10%

C: 7%

...

O: 1%

...

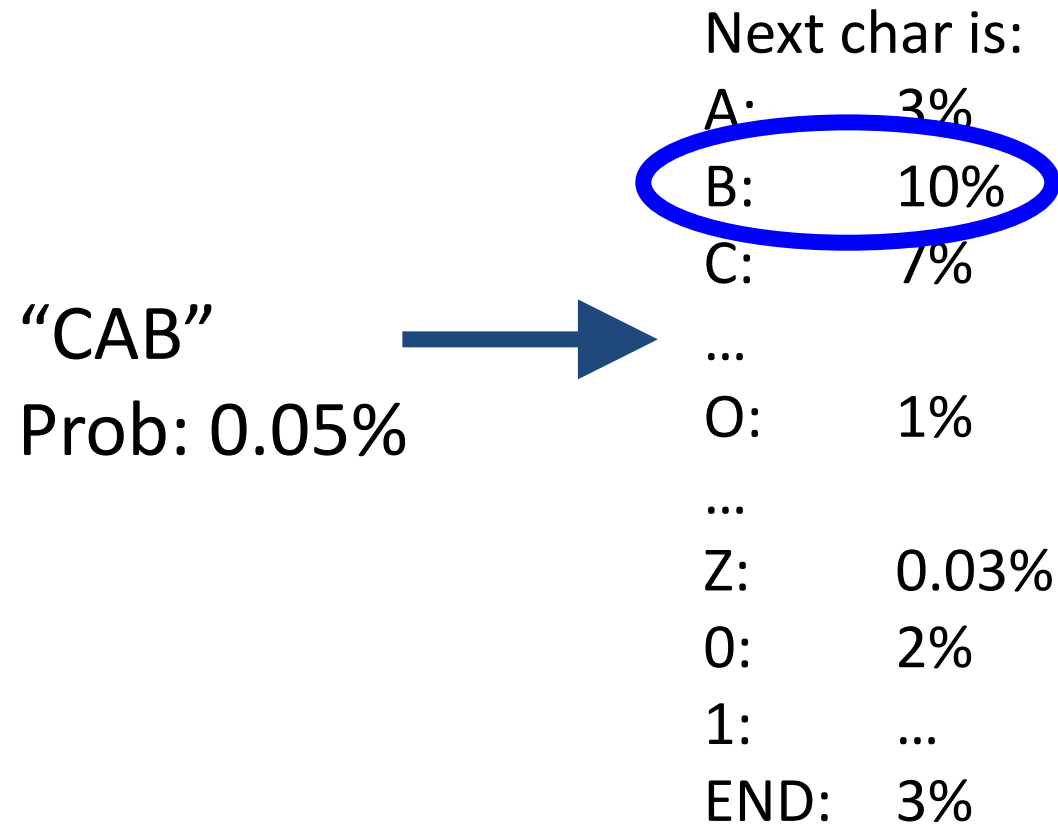
Z: 0.03%

0: 2%

1: ...

END: 12%

Generating Passwords



Generating Passwords

“CAB”

Prob: 0.05%



Next char is:

A: 4%

B: 3%

C: 1%

...

O: 2%

...

Z: 0.01%

0: 4%

1: ...

END: 12%

Generating Passwords

“CAB”

Prob: 0.05%



Next char is:

A: 4%

B: 3%

C: 1%

...

O: 2%

...

Z: 0.01%

0: 4%

1: ...

END: 12%

Generating Passwords

“CAB”

Prob: 0.006%

Generate in Descending Probability Order

CAB -	0.006%
CAC -	0.0042%
ADD1 -	0.002%
CODE -	0.0013%
...	

Design Space

- Model size: 3mb (browser) vs. 60mb (GPU)
- Transference learning
 - Novel password-composition policies
- Training data
 - Natural language
- (Many others)