L8: New Capabilities: Keyboard and Multiword Patterns & Dictionaries

Sudhir Aggarwal and Shiva Houshmand

Florida State University

Department of Computer Science

E-Crime Investigative Technologies Lab

Tallahassee, Florida 32306

August 5-7, 2015

Password Cracking
University of Jyväskylä
Summer School August 2015

Outline

- **Extensions**
  - Modeling Differences between Passwords
  - Keyboard Combinations
  - Better Identification of Alpha Strings
  - Developing Better Attack Dictionaries
  - LeetSpeak
- **Summary**

Extensions

- Modeling Differences between Passwords
- Keyboard Combinations
- Better Identification of Alpha Strings
- Developing Better Attack Dictionaries
- LeetSpeak

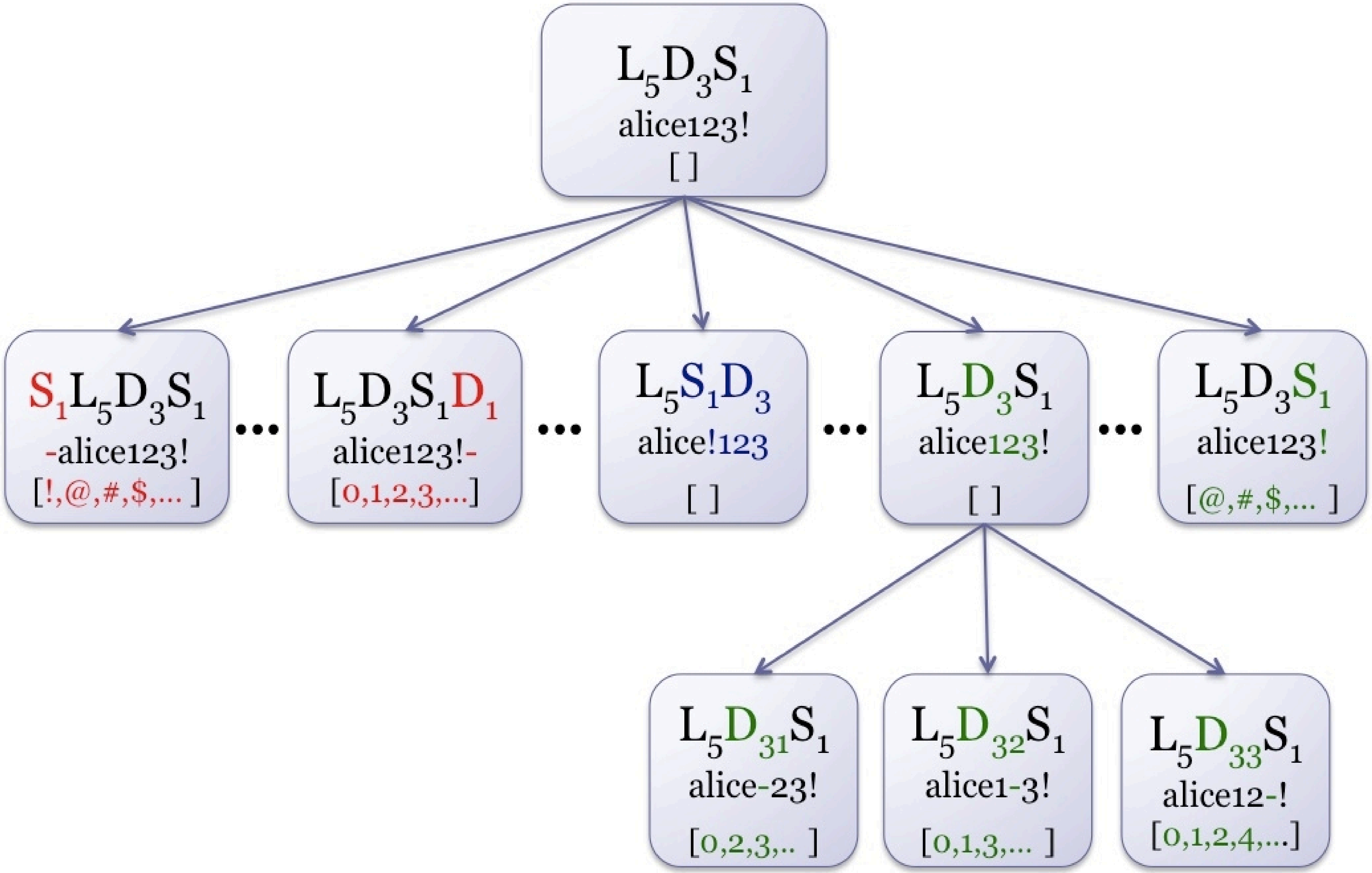
Modeling Differences: the problem

- **I know a user's password is alice123! and the user has changed this password. How do I make use of this information to crack the new password?**

- Try developing a conditional probability distribution. But, we do not have much data? And how does this help in defining a grammar?

- Try using Edit distance (Levenshtein distance) to find passwords close to the seed password. But how close is close?

- Try using transformational approach (s/1/2/, s/1/11/) where we use a set of regular expressions. Simple transformation seem ok but where do we draw the boundary?



Levenshtein Distance 1 Algorithm

What is the corresponding grammar for ***alice123!***?

Base	**Base Prob**	**Digits**	**Digits Prob**	**Symbols**	**Symbols Prob**
$L_5D_3S_1$	0.25	123	0.25	!	0.2
$L_5S_1D_3$	0.25	124	0.25	@	0.2
$L_5D_4S_1$	0.25	125	0.25	#	0.2
$L_5D_3S_2$	0.25	133	0.25	$	0.2
		12	0.5	%	0.2
		13	0.5	!!	0.33
		1234	0.5	!#	0.33
		1235	0.5	!@	0.33

How should I generate guesses?

- Use the edit 1 grammar. But I want to generate other guesses also. After all, the user might not have made small changes and might even have chosen a totally different password!
- This led us to the idea of merging probabilistic context free grammars. We can actually combine two different grammars and by extension any number of grammars!

The *Merge* of two grammars

- Let G_1 and G_2 be two probabilistic context-free grammars based on our structures of base structures and component structures. We construct a new grammar G_3 that we define as the *merge* of G_1 and G_2 and we represent it as:

$$G_3 = \alpha\, G_1 + (1\text{-}\ \alpha)\, G \qquad \text{where } 0 \le \alpha \le 1$$

- Consider a grammar rule R in G_1 or $G_{2.}$ Let the probability of R in G_1 be r_1 and the probability of R in G_2 be r_2. (Note that if R is not in a grammar its probability is viewed as 0.) Then the probability r_3 of R in G_3 is:

$$r_3 = \alpha\, r_1 + (1 - \alpha)\, r_2$$

$L_5D_3S_1$	0.25
$L_5S_1D_3$	0.25
$L_5D_4S_1$	0.25
$L_5D_3S_2$	0.25
123	0.25
124	0.25
125	0.25
133	0.25
12	0.5
13	0.5
1234	0.5
1235	0.5
!	0.2
@	0.2
#	0.2
$	0.2
%	0.2
!!	0.33
!#	0.33
!@	0.33

Edit 1 Grammar
$W_1 = 0.8$

+

$L_4D_2S_1$	0.5
$L_3D_3S_2$	0.3
$L_5D_3S_1$	0.07
$L_6D_4S_2$	0.05
$L_8D_2S_1$	0.05
$L_5D_3S_2$	0.03
999	0.6
111	0.3
123	0.1
88	0.5
11	0.5
5656	0.5
1234	0.3
0909	0.2
!	0.4
)	0.3
?	0.2
%	0.1
!!	0.3
##	0.3
$#	0.2
!#	0.2

Initial Grammar
$W_2 = 0.2$

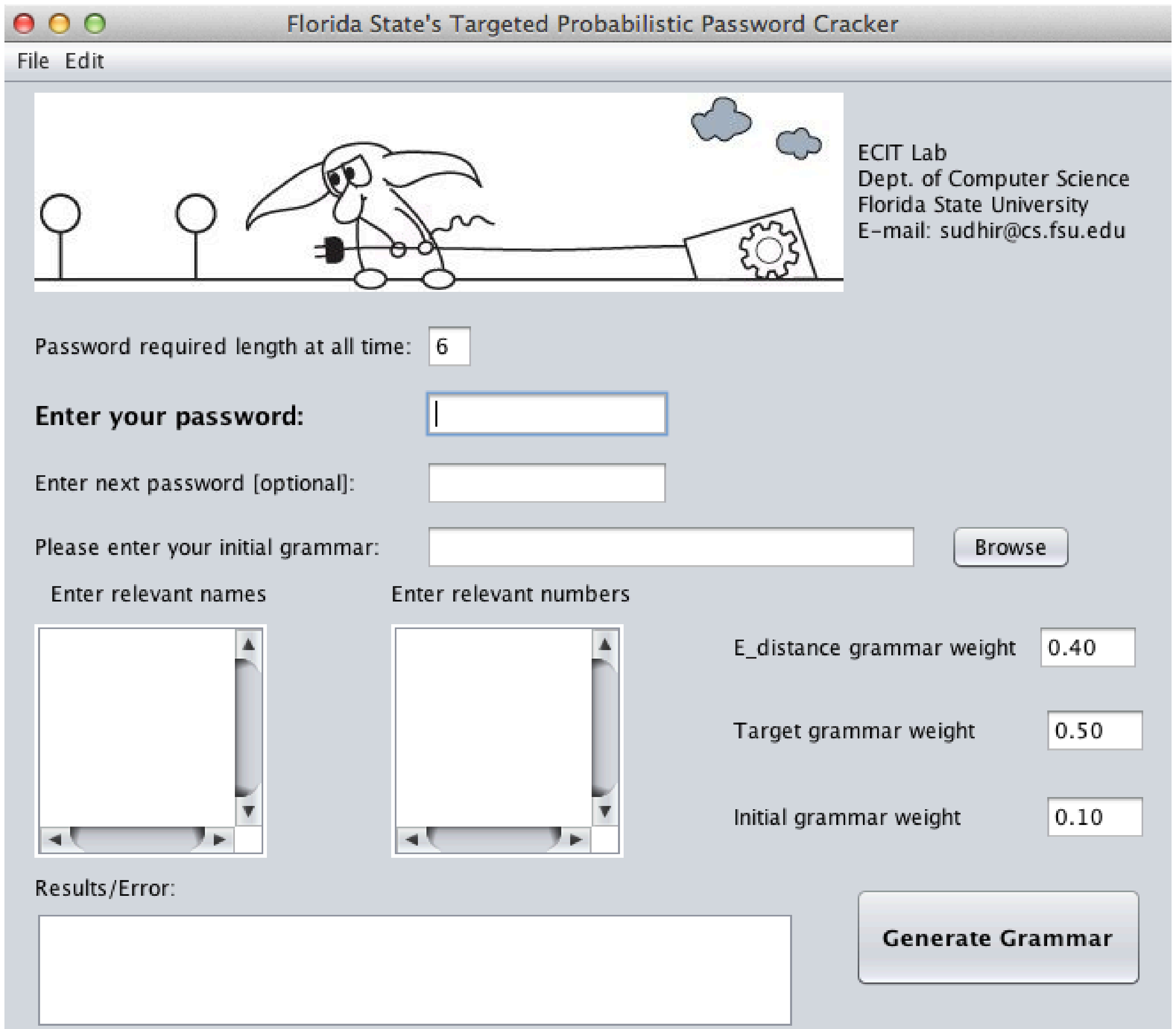
=

$L_5D_3S_1$	0.214
$L_5D_3S_2$	0.206
$L_5D_4S_1$	0.2
$L_5S_1D_3$	0.2
$L_4D_2S_1$	0.1
$L_3D_3S_2$	0.06
$L_6D_4S_2$	0.01
$L_8D_2S_1$	0.01
123	0.22
124	0.2
125	0.2
133	0.2
999	0.12
111	0.06
12	0.4
13	0.4
88	0.1
11	0.1
1234	0.46
1235	0.4
5656	0.1
0909	0.04
!	0.24
%	0.18
#	0.16
$	0.16
@	0.16
)	0.06
?	0.04
!!	0.324
!#	0.304
!@	0.264
##	0.06
$#	0.04

Additional Research Directions Explored

- We now handle keyboard combinations and multiwords when we want to consider edit distance changes given a previous password
- We also consider semantic transformations to entities such as dates incorporating possible variations
- We are gathering data on developing attacks given a password and a changed one. This is through a series of surveys we have been conducting

Demo Modeling Differences



Old password1	Old password2	New password	Number of Guesses made to crack	Merged Or Edit distance grammar
russell	-	RUSSELL	1	Edit distance
russell1	-	russell	1	Edit distance
abc2009	-	pm2009	4,334,388	Merged
maverick	-	maverick7	118	Edit distance
dreamhope	-	hopehope	-	Merged
hopeful	-	hopeful1	14	Edit distance
starwars	-	starwars1	17	Edit distance
sweetie	-	sweetie1	20	Edit distance
krishna	-	krishnap	-	Merged
hope77	-	hope22	2,111	Merged
bland0608	-	plat0608	136,066,042	Merged
milena	-	Milena	4	Edit distance
milena	-	milene	-	Edit distance
bluemoon1	bluemoon2	bluemoon3	1	Edit distance
moonlight	-	redmoonlight	-	Merged
1writer	-	writer	1	Edit distance
1blackcat	-	blackcat	1	Edit distance
starwars	starwars5	starwars55	1	Edit distance
sweety	-	SWEETY	308	Merged
groove5721	-	Katie5721	-	Merged
171995	-	may171995	47,881,797	Merged
skymoon7	-	moon7sky	-	Merged
chomsky$po	-	po$chomsky	-	Merged
gamegreen	-	greendoc	-	Merged
d30023286	-	30023286	1	Edit distance
081983lori	-	081983	1	Edit distance
243currier	-	24378443	-	Merged
19632439	-	19632007	-	Merged
blackhawk	-	black7out	-	Merged

Extensions

- Modeling Differences between Passwords
- Keyboard Combinations
- Better Identification of Alpha Strings
- Developing Better Attack Dictionaries
- LeetSpeak

Modeling Keyboard Combinations

- What are keyboard combinations? How can we define them?
- How useful are keyboard combinations
- How do we train for them
- How do we use them in cracking

What is a Keyboard Pattern?

Classic example is “querty”
Intuitive idea is that that it is a shape on the keyboard
How do we define these shapes
How complex a model makes sense
Contiguity of characters is important

What is a shape?

u v
l c r
d e

qwerty: (q) rrrrr
zsdfvcs: (z) vrrell
1111222334: (1) cccrcccrcr
Limited patterns to length 3 but allowed any case
Decided not to consider shapes which required skipping some keys

Keyboard shapes and patterns

Shapes

Shapes	Probability
rrrrr	0.261
ccccc	0.146
uceuc	0.038
lcrlc	0.024
ueueu	0.016
rlrlr	0.015
rclrc	0.014
eveve	0.013

Patterns

Patterns	Probability
qwerty	0.182
asdfgh	0.036
aaaaaa	0.029
deedee	0.023
poopoo	0.019
zxcvbn	0.016
xxxxxx	0.014
1q2w3e	0.009

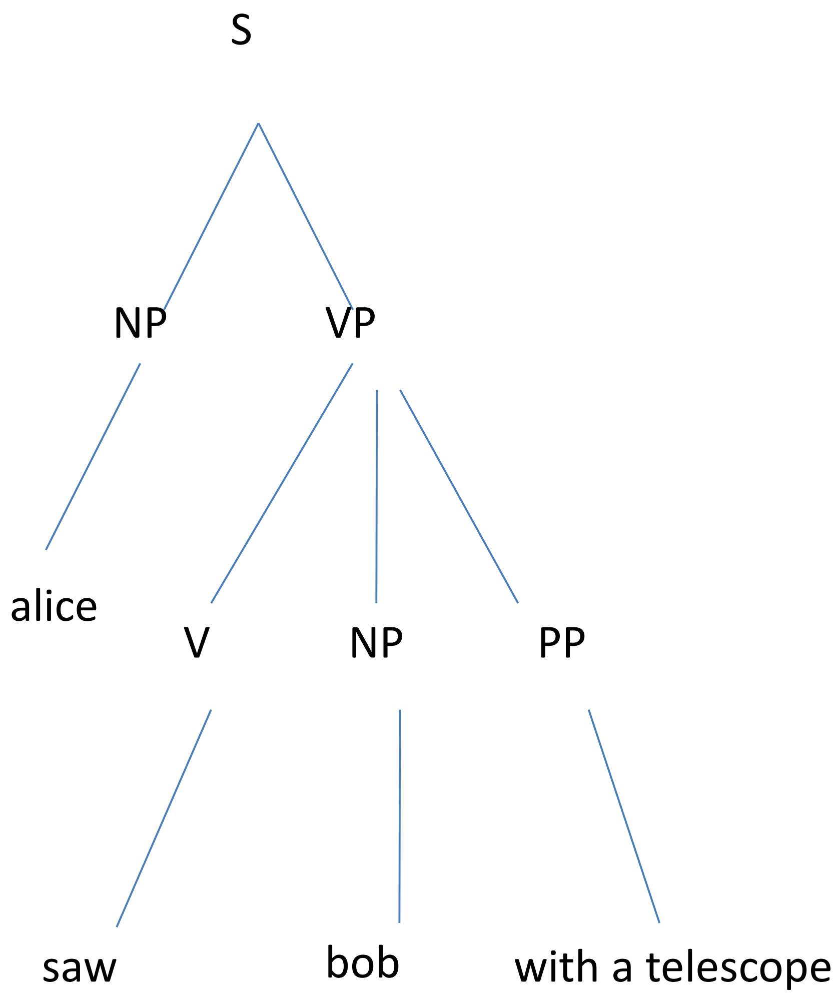
Keyboard Combinations: Ambiguity

- Keyboard combinations are physical combinations taken from the keyboard such as qwerty

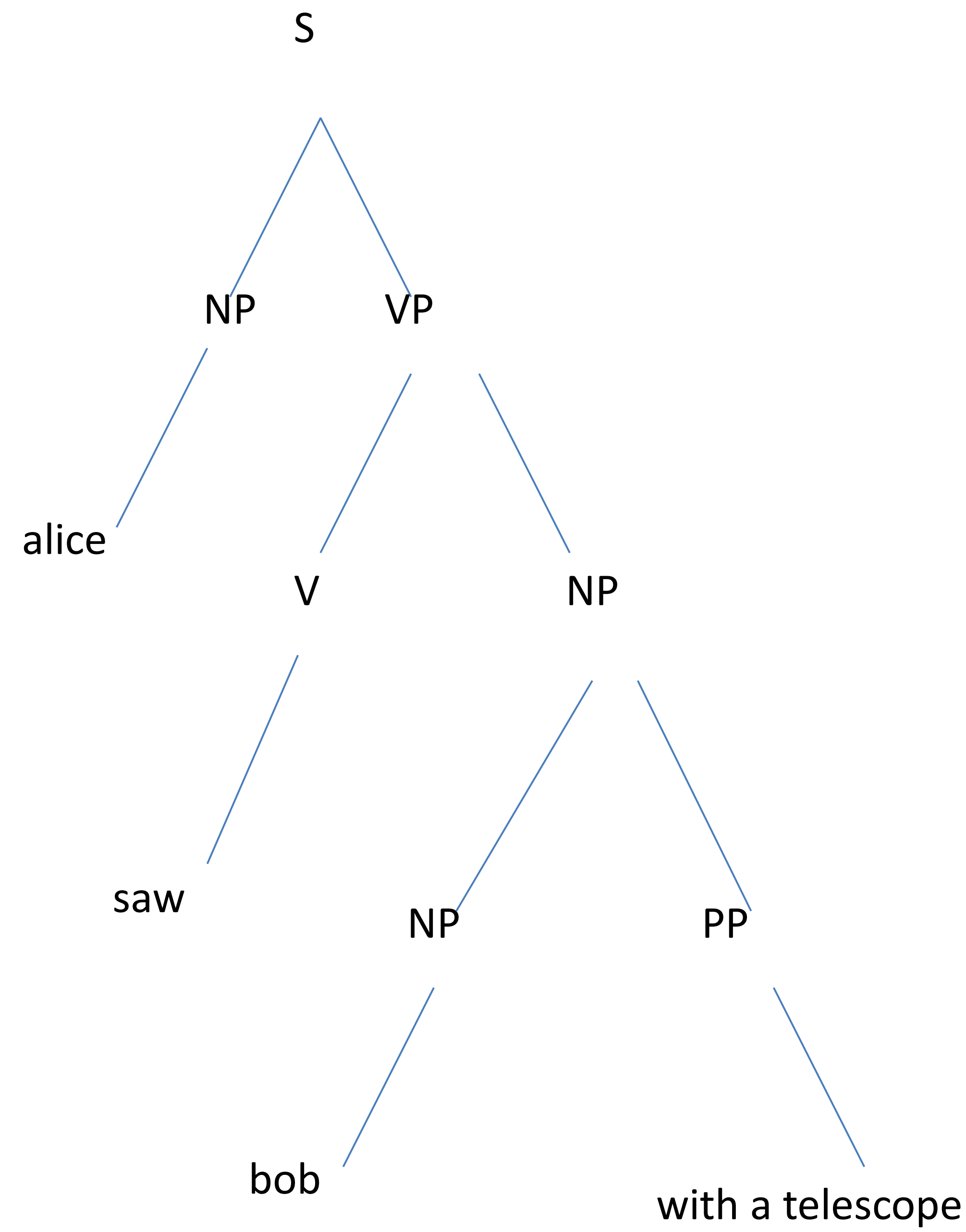
- Should we handle ambiguous grammars? Can the same string be derived by two different parses

  - This becomes a problem because the probability of each parse must to summed to get the final probability. Eg. *23were* is both K_6 and D_2L_4.

- Should we include keyboard combinations in the dictionaries? Then this is not part of the grammar.



Derivation tree 1



Derivation tree 2

Problems with Ambiguity

- The problem of ambiguity is that is we have two parse trees that generate the same terminal string with probabilities p_1 and p_2, the probability of the terminal string is the sum of these. So how do we generate in highest probability order?

- Furthermore suppose we have alice1234. Is the 1234 a digit string D4 or a keyboard pattern K4? Also do we really care?? And can we tell what the password author intended?

- For example, if we have base structures L_5D_4 or L_5K_4 we would eventually generate either one. Does it makes sense to worry about what was intended?

Decisions about Ambiguity

- The first rule is that if a substructure is purely digits or purely special symbols, we will classify it as $\mathbf{D_i}$ or $\mathbf{S_i}$.
- The second rule is that any substring of at least 3 characters in length that does not fall under the first rule will be classified as a $\mathbf{K_i}$ if it is a keyboard pattern and is of maximal length. For example *e4e458* would be $\mathbf{K_5D_1}$ as the maximal length keyboard substring must be used.

Modifying the Grammar: K structures

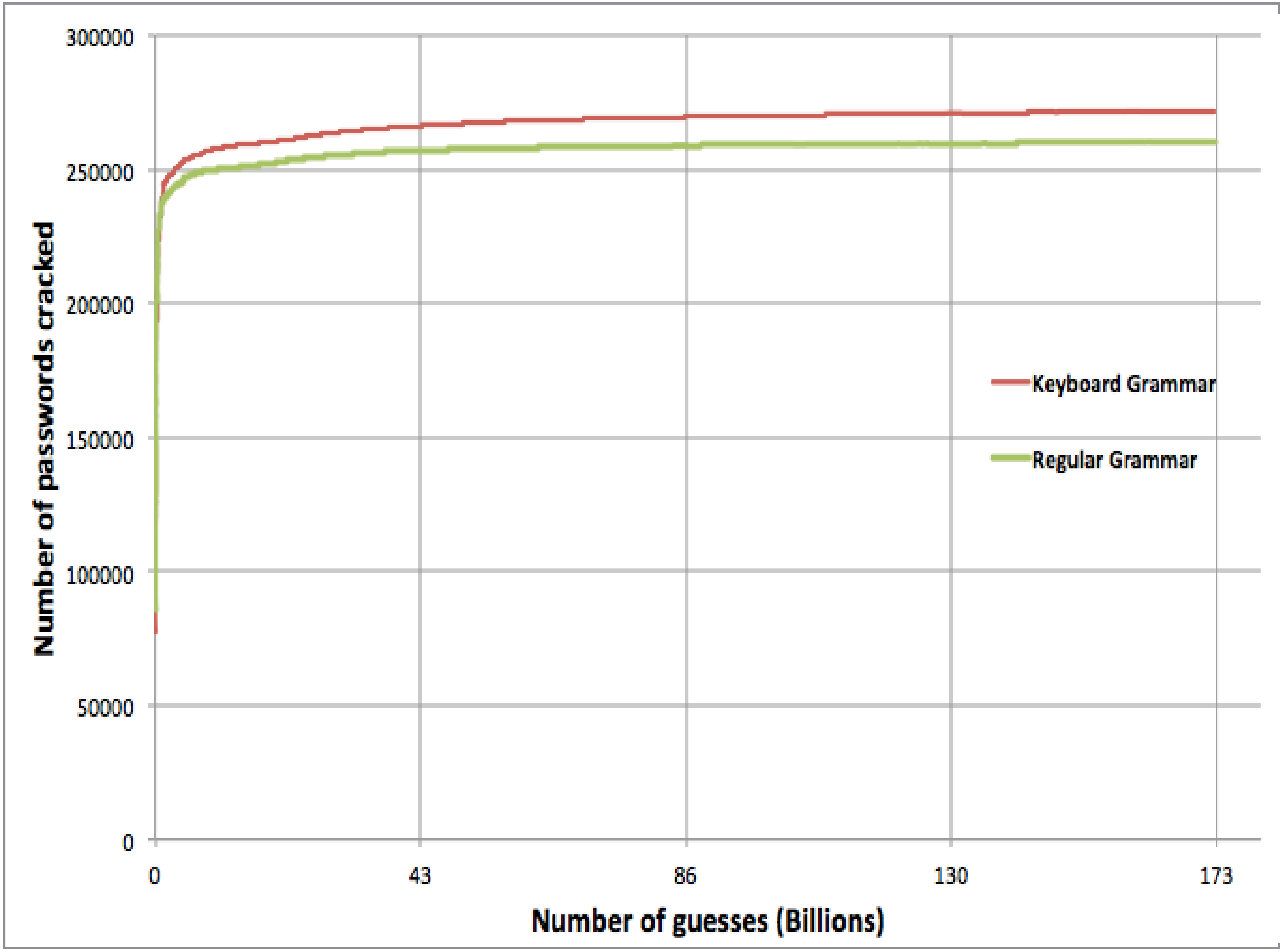
Password	Original	Keyboard
asdf	L_4	K_4
q1q1	$L_1D_1L_1D_1$	K_4
ASD1234QW	$L_3D_4L_2$	$K_3D_4L_2$
$%^&	S_4	S_4
qaz12zaq	$L_3D_2L_3$	$K_3D_2K_3$
q1!2	$L_1D_1S_1D_1$	K_4

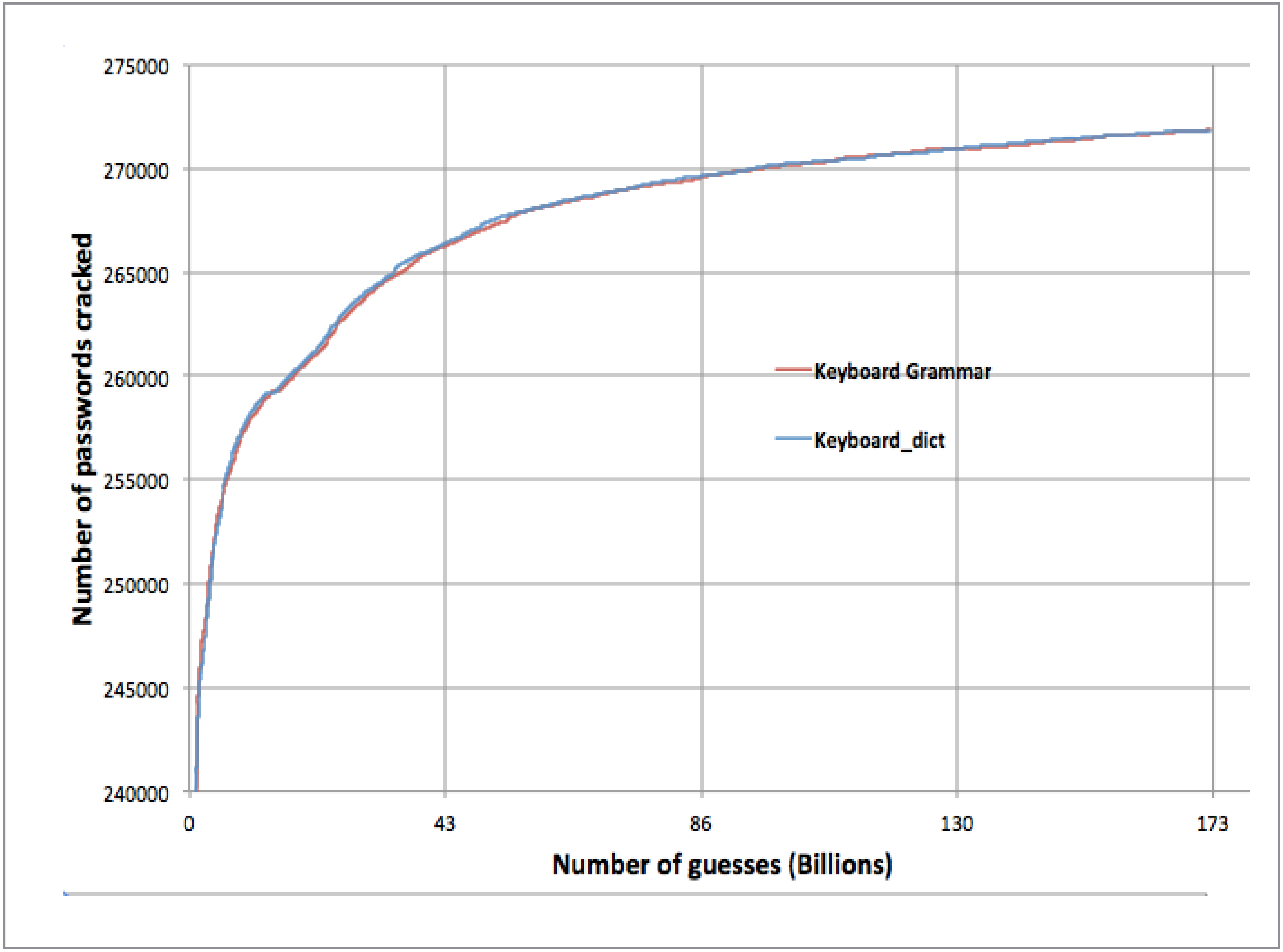
A Problem with the Decision

- Note that “5querty” certainly has a keyboard pattern. But “1sees” is not so clear that it is a D_1K_4.
- In the first case we know that querty is not really a word (although for the specific choice that could be argued!) but in the second case it seems more likely that it is a word.
- So we decided to find a way to experiment with these choices: we introduced the notion of a *training dictionary* that could help us decide.

Training Dictionary

- While training and looking for patterns detect a keyboard pattern such as “were” and treat it as if it was an L structure and not a K structure
- We can filter out such K patterns with the training dictionary
- It turns out that a training dictionary also has many other uses
- We sometimes call the dictionary used in cracking an attack dictionary to clearly distinguish it from the training dictionary if necessary


300000
250000
200000
150000
100000
50000
0
Number of passwords cracked
0
43
86
130
173
Number of guesses (Billions)
Keyboard Grammar
Regular Grammar



275000
270000
265000
260000
255000
250000
245000
240000
Number of passwords cracked
0
43
86
130
173
Number of guesses (Billions)
Keyboard Grammar
Keyboard_dict

Smoothing Keyboard Patterns

- We can find keyboard patterns as we defined with or without using our training set.
- Suppose however we want to try keyboard patterns that we did not find in the training set.
- Just as we did for digits, we decided to smooth over keyboard patterns. But how should we do this.
- We decided to smooth based only on the *shapes* we found. Furthermore we adjust the smoothing based on the probability of the shapes encountered.
- This was a reasonable compromise between smoothing everything and no smoothing at all.

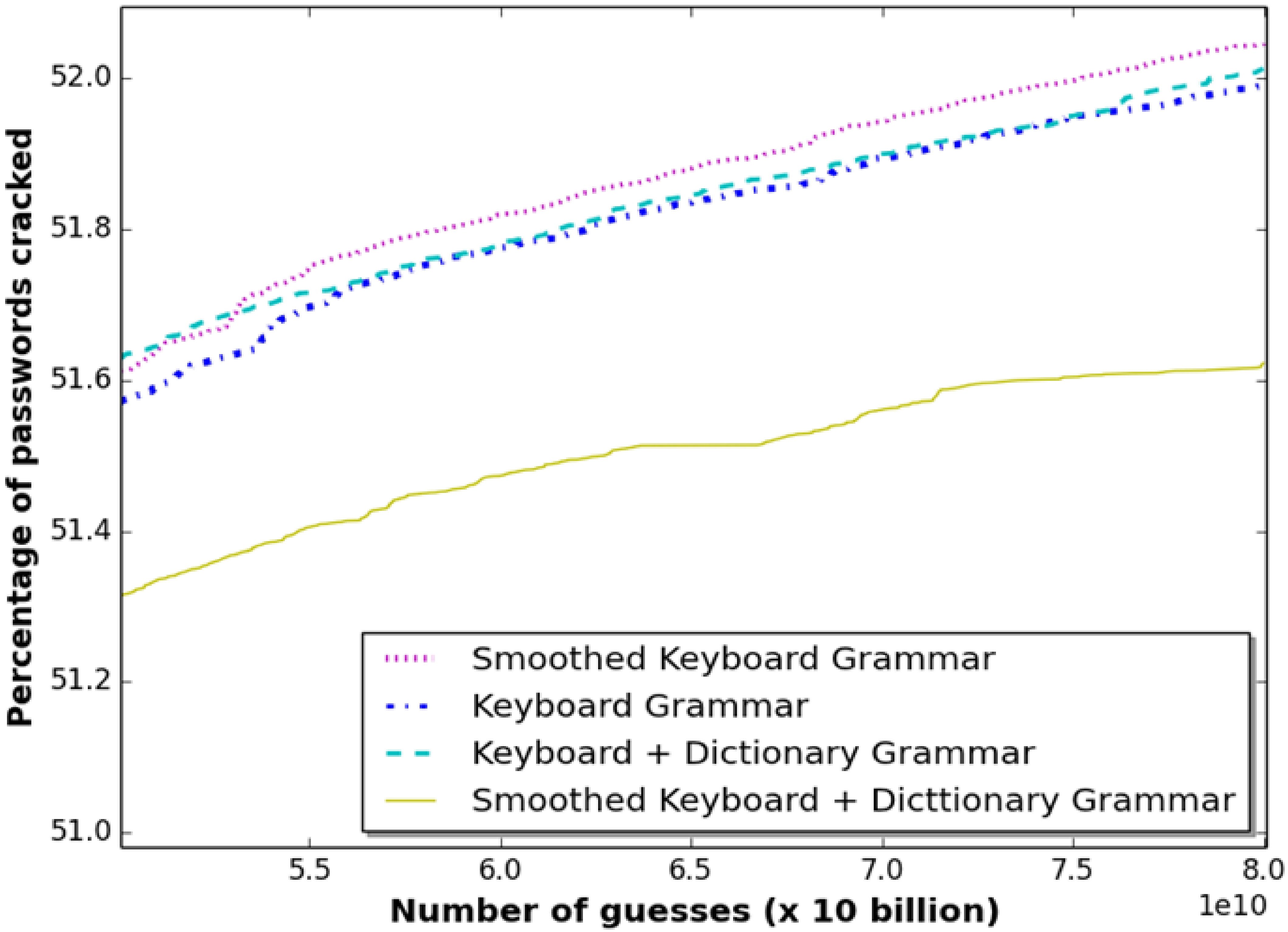
Smoothing Implementation

$$\boldsymbol{Prob(pattern) = Prob(s)\ (N_i + \alpha)\ /\ (\Sigma\ N_i + C\ \alpha)}$$

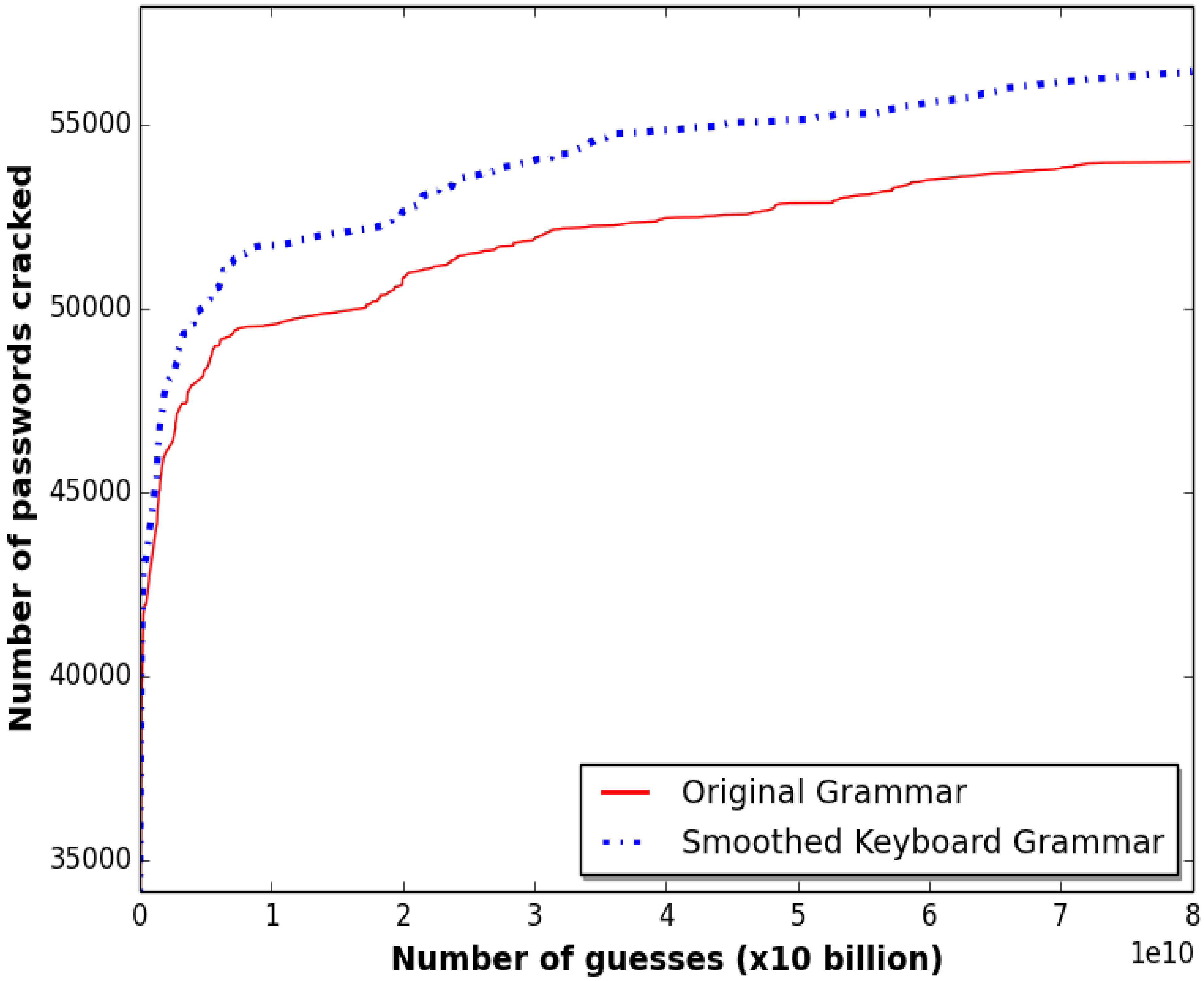
- *(pattern(i, s)) = pattern is the ith keyboard pattern of shape s.*
- *Prob*(*s*) is the probability of the keyboard shape *s* (such as $\boldsymbol{r^5}$) given the length of the keyboard pattern
- N_i is the number of times the ith keyboard pattern (of this shape) was found
- α is the smoothing value
- $\Sigma\ N_i$ is the sum of counts of the patterns found for shape *s*
- *C* is the total number of unique patterns for this shape.

Experiments: Combined-set

- Combined Several lists: Size of training set
  - RockYou – 0.5 million
  - Myspace – 31 thousand
  - Hotmail – 5 thousand
- A similar (independent) set used for cracking



Results using Combined-set



CSDN-set: Chinese language forum site

Extensions

- Modeling Differences between Passwords
- Keyboard Combinations
- Better Identification of Alpha Strings
- Developing Better Attack Dictionaries
- LeetSpeak

L- component in Base Structures

- We have previous simply replaced the L component by a dictionary word of the relevant length
- What kinds of patterns can we find in the L – structures?
- What patterns are useful?
- Note that we have already defined keyboard patterns which involve L – structures but also other structures
- Should we focus only on the L –component part?

Initial Focus

1. Dictionary words
2. Double dictionary words
3. Double patterns
4. Other

What are we missing? Note that we decided to look only at patterns within only a specific L – structure but not spanning beyond that.

Classification of Alpha Strings: A-structures

Classification	**Example**
Dictionary Word -- L	password
Double dictionary word --R	boatboat
Double pattern -- R	xyzxyz
Multiword -- X	Iloveyou
Other -- L	ahskdi

Further Understanding Alpha Strings

- Let's look at the Combined Data Set
  - It has a bit over 500,000 passwords, so it is pretty big
  - These are the top 5 most probable base structures
  - It turns out Multiwords are very common

Base Structure	Dictionary	Multiwords	Double Dictionary
L_6	38.47%	22.63%	1.92%
L_7	32.85%	31.52%	0.00%
L_8	22.51%	38.17%	1.29%
D_6	N/A	N/A	N/A
L_9	14.33%	46.36%	0.00%

Finding Multiwords

- Many issues arise in determining if an L structure is a multiword
    - How do we develop an algorithm to break up the multiwords
    - How do we use a training dictionary
    - How efficient are the algorithms
    - How effective are the algorithms
    - Possibly several choices in the break
    - It turns out that this problem, called “word breaking or word segmentation” has been studied in other contexts

Algorithms Explored & Issues

- Special algorithms to break up the A – string into two or three words. (Find the first word, starting from the left (or right or both) and check the remainder
- Give preference to breaks that have fewer words
- Recursive algorithms that break words from the left or right.
- Finding all break ups versus only one breakup
- Scoring function to choose among breakups
- What kind of training dictionary to use for finding breakups – that is what are appropriate component words

Alternative Reductions

String	Alternative Interpretations
americarules	america rules, am eric a rules
gotohell	go to hell, got oh ell
woodstock	woods tock, wood stock
hairspray	hair spray, hairs pray
ladiesman	ladies man, la dies man
Thisisit	This is it, this i sit

Adding New Variables to the Grammar

L	Letter (used for Dictionary Words and *Other*)
D	Digit
S	Symbol
K	Keyboard Pattern
X	Multiword
R	Repeated (used for *double* words and *double* patterns)

Deriving the grammar: single level approach

- From the start symbol, directly get new base structures using the new variables.

$S \rightarrow R_8D_3$

$S \rightarrow L_8D_2$

$S \rightarrow X_8S_1$

$S \rightarrow R_8D_3 \rightarrow \text{boatboat}D_3 \rightarrow \textbf{boatboat123}$

$S \rightarrow L_8D_2 \rightarrow \text{password}D_2 \rightarrow \text{password11}$

$S \rightarrow X_8S_1 \rightarrow \text{johnmary}S_1 \rightarrow \text{johnmary\#}$

Deriving the grammar: two level approach

- From the start symbol, derive an **A** structure, then get the new base structures using the new variables

$$\boldsymbol{S \rightarrow A_8D_3} \qquad \boldsymbol{A_8 \rightarrow R_8}$$
$$\boldsymbol{S \rightarrow A_8D_2} \qquad \boldsymbol{A_8 \rightarrow L_8}$$
$$\boldsymbol{S \rightarrow A_8S_1} \qquad \boldsymbol{A_8 \rightarrow X_8}$$

$$\boldsymbol{S \rightarrow A_8D_3 \rightarrow R_8D_3 \rightarrow \text{boatboat}D_3 \rightarrow \text{boatboat123}}$$
$$\boldsymbol{S \rightarrow A_8D_2 \rightarrow L_8D_2 \rightarrow \text{password}D_2 \rightarrow \text{password11}}$$
$$\boldsymbol{S \rightarrow A_8S_1 \rightarrow X_8S_1 \rightarrow \text{johnmary}S_1 \rightarrow \text{johnmary\#}}$$

Effect of the Choices

- The probabilities in the two approaches would not be the same
- The training is different: The two level approach gives many more base structures which can be good but in some pathological cases is a real problem
- We have basically implemented the two level approach but not in an obvious was and the resulting files look as before but with the new variables
- Pathological example:

aa1aa2aa3aa4aa5aa6aa7aa8aa9

Creating “Ground Truth” for multiwords

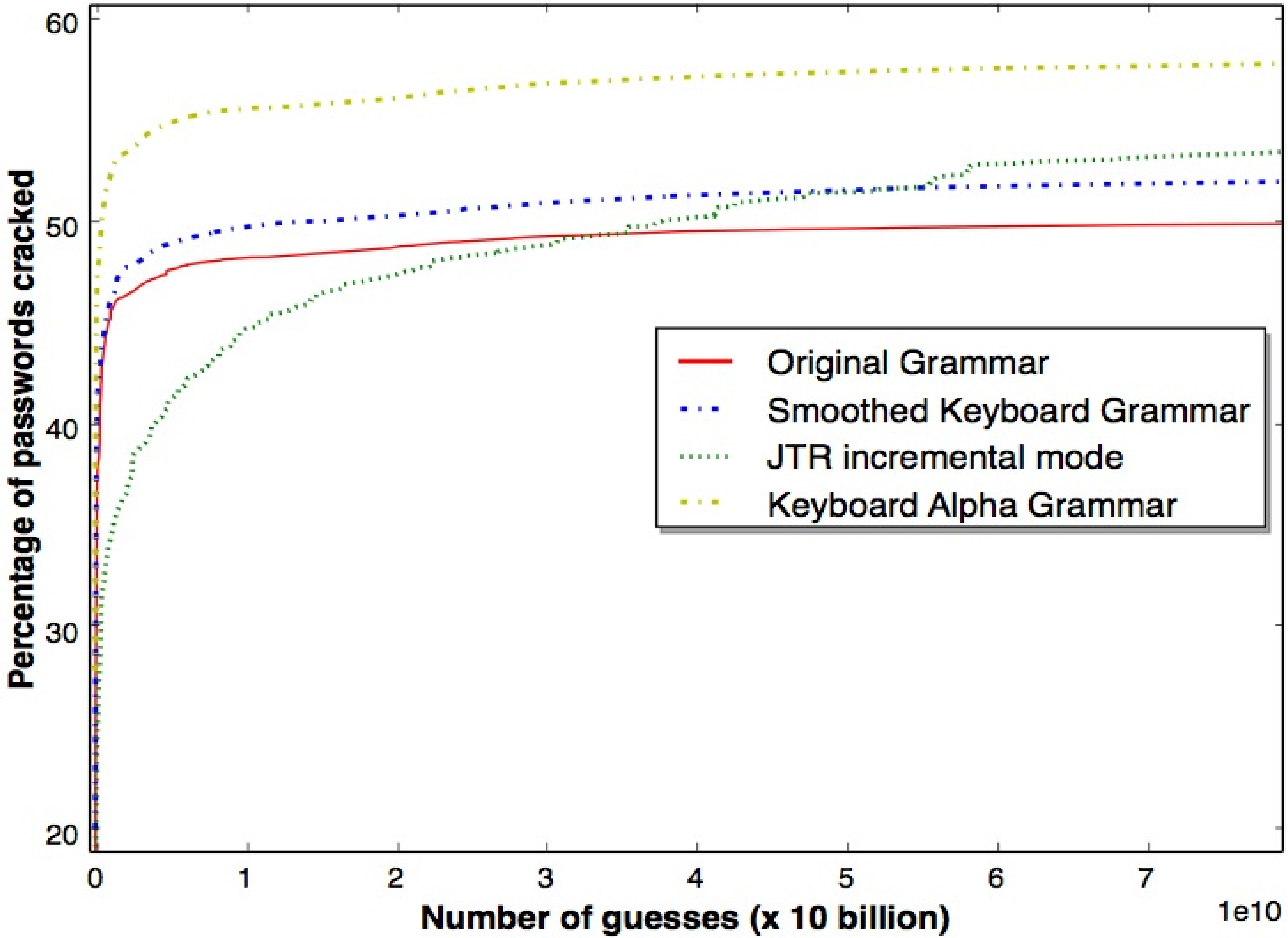
Breakdown	Agreement	Comments
pr.inc	Not a multiword	Shortened “prince”?
i.love.you	Best breakdown	
let.mein	Not best breakdown	let.me.in
a.ms	Not a multiword	
em.in.em	Not a multtiword	name
sair.ram	Not a multiword	Hindi name
a.did.as	Not a multiword	Sports brand
parol.a	Not a multiword	Spanish word
mo.mph.ali	Not a multiword	Hindi word

Modifications to cracking system: R Structures

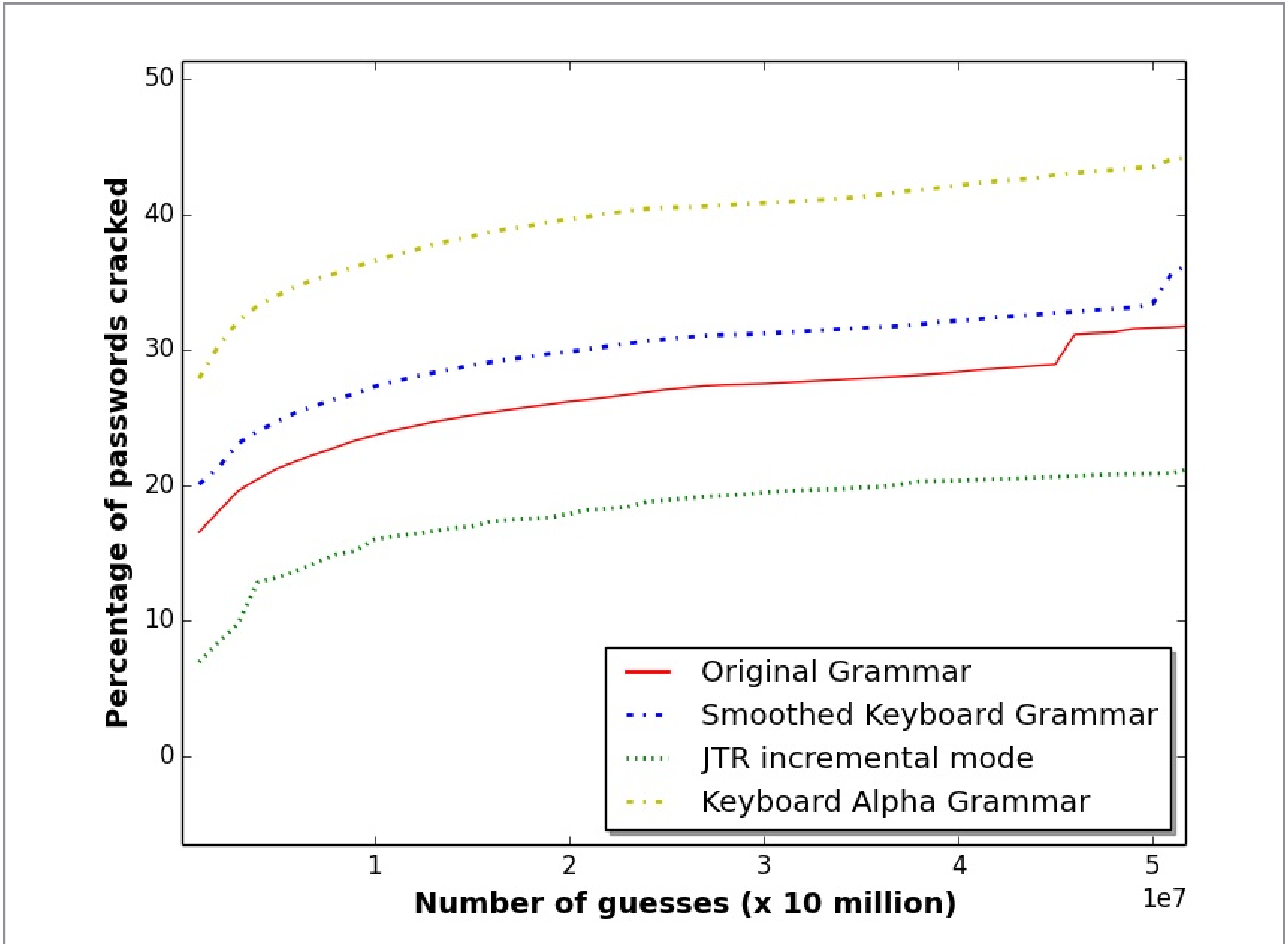
- **Handling the new R structure**
  - Similar to L structures, these are derived from a dictionary
  - Essentially, when we read in the dictionary, we create a double word dictionary with the same probabilities as the single word dictionary
  - Substituting for an R – structure thus is done using a container that has all double words of the specific length and probability class.
  - Note that the probability of a base structure with the R structure is learned as before and that both double word and double pattern are treated the same way

Modifications to cracking system: X Structures

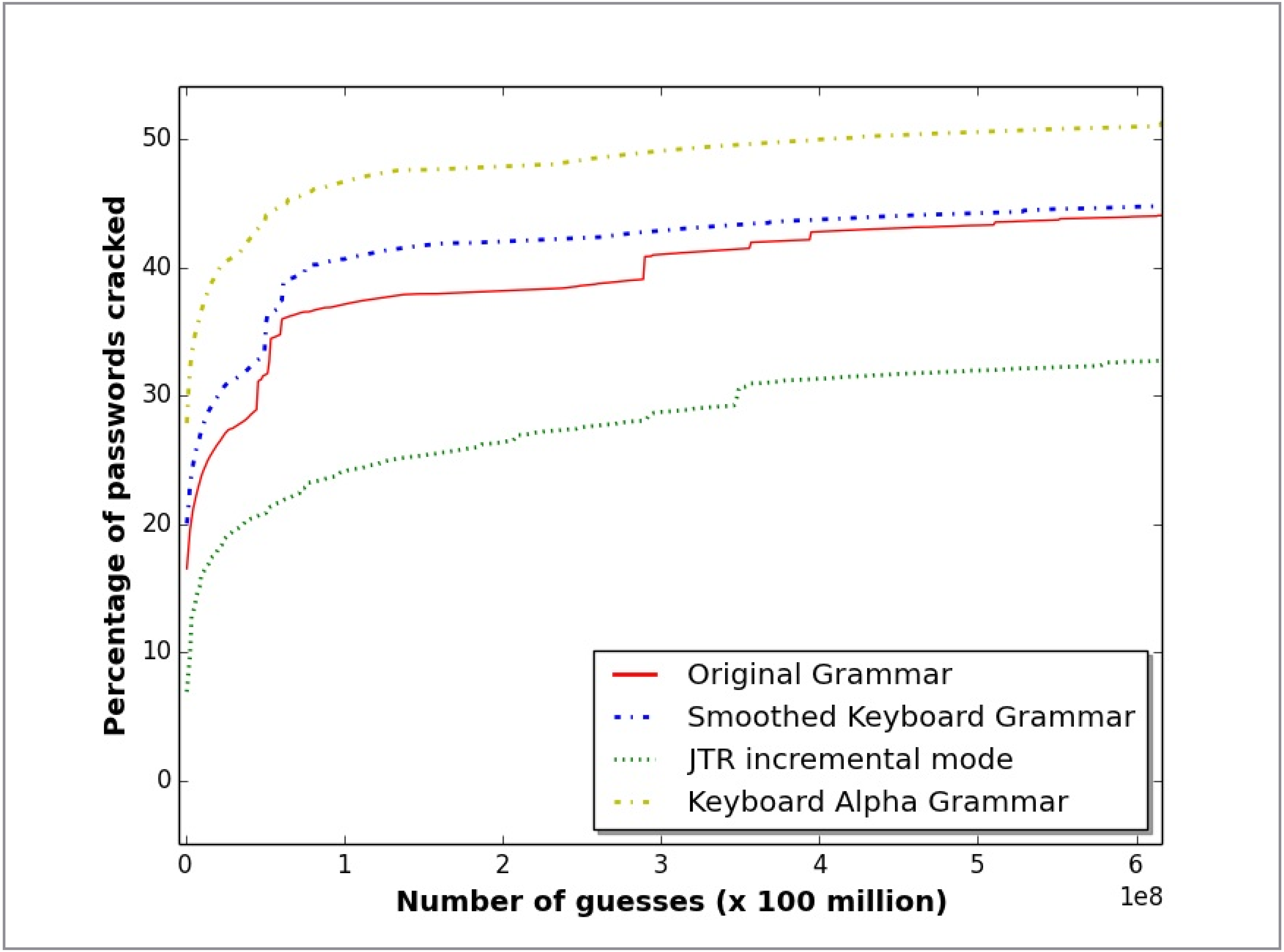
- **Handling the new X structure**
  - Multiwords
  - Similar to Keyboards, Digits and Symbols
  - Find multiwords by length: X_n
  - Assign probabilities to the various multiwords found
  - For multiwords, we do not do smoothing at this time



Results with Combined-set



Percentage of passwords cracked
Number of guesses (x 10 million)
1e7
0
10
20
30
40
50
1
2
3
4
5
Original Grammar
Smoothed Keyboard Grammar
JTR incremental mode
Keyboard Alpha Grammar



Percentage of passwords cracked
Number of guesses (x 100 million)
1e8
0
10
20
30
40
50
0
1
2
3
4
5
6
Original Grammar
Smoothed Keyboard Grammar
JTR incremental mode
Keyboard Alpha Grammar

Extensions

- Modeling Differences between Passwords
- Keyboard Combinations
- Better Identification of Alpha Strings
- Developing Better Attack Dictionaries
- LeetSpeak

Attack Dictionaries

- There are many different ways that the term “dictionary” has been used in password cracking so it is important to be sure how it is used in any specific context.
  - It could be the set of guesses themselves
  - It could be as source of passwords as well as a base for applying mangling rules
  - It could be as a language based collection of words
  - It could be a as some other collection of items
  - Our use is as a source of replacements for our alpha strings and the entries are generally words in a language

Multiple Dictionaries in PPC

- Probabilities can be assigned to dictionaries. These are actually indicated as relative weights for the dictionaries in the command line.
- Suppose a dictionary has $|L_i| = n_i$ words of length *i*. Then the probability of each L_i word is $1/n_i$. Note that if the fewer the number of words, the greater is the probability of each word.
- When using multiple dictionaries the weights of words of structures L_i that occur in multiple dictionaries increases by a complex formula based on the dictionary weights and the word weights.
- Essentially, we divide the set of words of length *i* into a number of classes (the same as the number of dictionaries) with each class having elements of the same probability. The total probability of all words of length *i* is 1.
- This can be viewed generating a set of containers for each for each L structure.

Comparing Attack Dictionaries

- Attack dictionaries have been traditionally created in a very ad-hoc manner
- Important wordlists of previously broken passwords (golden dictionary) may be added
- Different sized dictionary of words in different languages can be used, etc.
- *Is there any way to measure the effectiveness of a particular dictionary?*

How to measure effectiveness?

- How can we measure the effectiveness of a dictionary of words W? Let the words be $\{w_1 \ldots w_n\}$.
- We developed the notion of coverage and precision with respect to a reference set of passwords R
- A word is found in R, with I(w, R) = 1, if w is found in some L-structure of a password in R else I(w, R) = 0.
- The count C(w, p) of a password that has k A-structures and c instances of w is c/k
- Let R_L be the subset of R that have a least 1 A-structure

Coverage and Precision Definitions

$$C(W,R) = \frac{1}{|R_L|}\sum_{i=1}^{n} C(w_i,R)$$

$$P(W,R) = \frac{1}{|W|}\sum_{i=1}^{n} I(w_i,R)$$

Coverage, Precision and Perfect Dictionary

- Coverage measures how useful the words in the dictionary are for cracking the passwords in the reference set.
- For an ideal coverage of 1, every word in an A-structure of the reference set R would be a word in the target dictionary.
- We define a perfect dictionary (D_R) as a dictionary that has exactly those words found in R. Note that the perfect dictionary has both coverage and precision equal to 1.

Passwords sets in the Experiments

- Combined-training: ½ million Rockyou, 31 K MySpace, 5 K Hotmail
- Combined-test: same numbers as combined-training but excludes any passwords chosen for combined-training.
- Yahoo-test: 143 K from Yahoo set.
- Rockyou-test: 143 K from Rockyou set (different passwords from before)

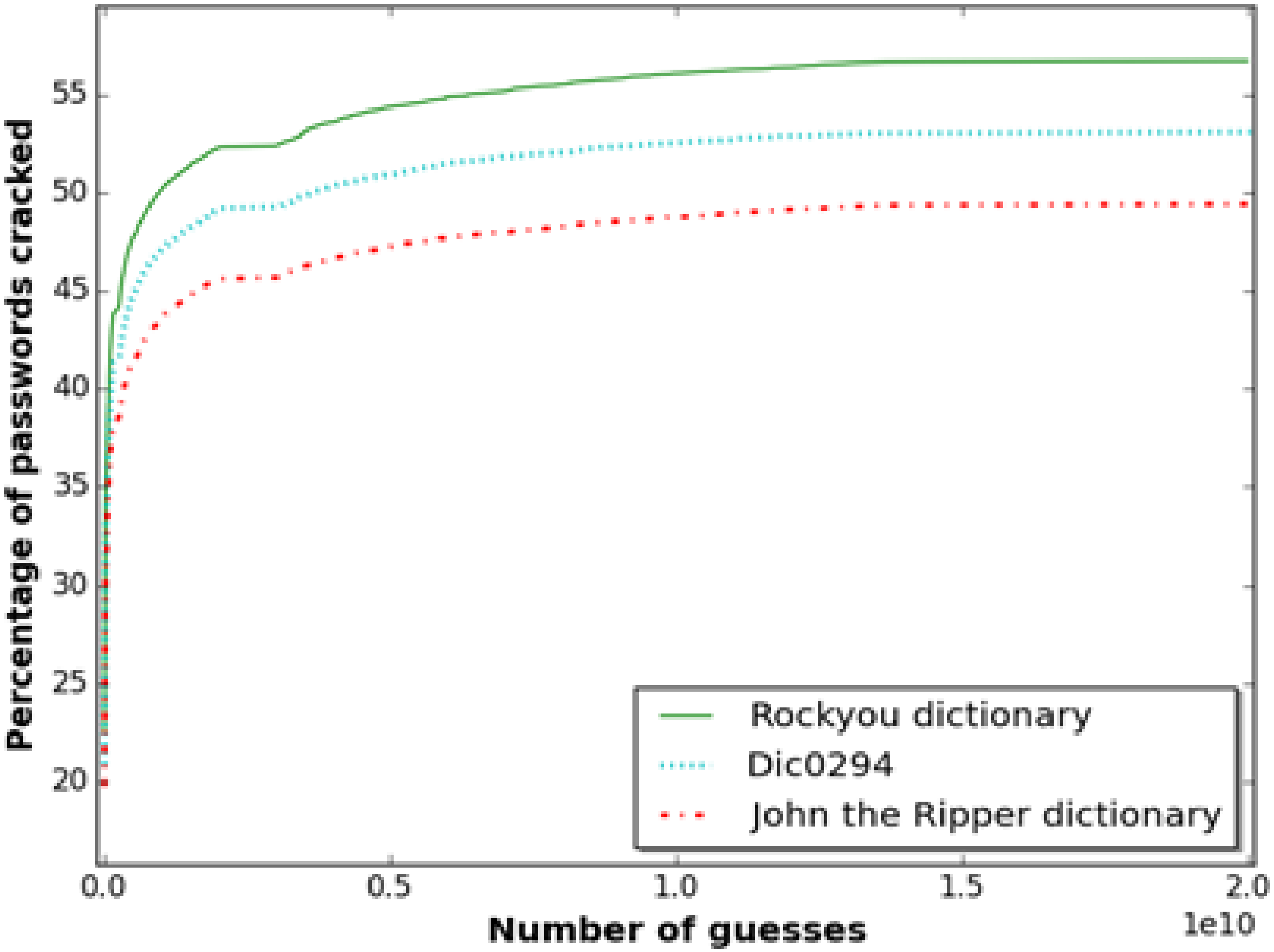
Base Dictionaries in the Experiments

- **Dic0294:** Often used in password cracking. Note that digits and special symbols have been removed from the original Dic0294. **Size 728K**.
- **JtR_eng Dict:** Created a similar sized dictionary from JtR wordlist collection. **Size 728K**.
- Rockyou Dict: Created a similar sized dictionary from 2.5 million Rockyou set by eliminating duplicates when stripping out the words in the A-structures. **Size 728K.**

Dictionaries with reference set Combined-test. Calculating Coverage and Precision

Dictionary	Size	Coverage	Precision
Rockyou dict	728,376	0.74	0.11
dic0294	728,216	0.55	0.06
Jtr_En dict	728,749	0.49	0.05

Cracking Yahoo-test



Improving Dictionaries

- Goal: systematically improve a given dictionary
  - Start with baseline dic0294 – improve Coverage and or Precision
  - First explored improving coverage while keeping Precision fixed
  - Then explored improving precision while keeping coverage fixed

Improving Coverage wrt Reference Combined-test

- Let D be baseline dic0294 with (C, P) = (0.55, 0.06). Let ct be the reference set combined-test. Let D_{ct} be the perfect dictionary for the reference set.

  - Add n_r words from D_{ct} (in highest coverage order) to D. In order to maintain precision P also add n_n words not in D_{ct} to D.

  - Created dic0294_c70 and dic0294_c90 (P= 0.06)

  - *Can you figure out precisely how and how many words to add?*

Improving Precision wrt Reference Combined-test

- Let D be baseline dic0294 with (C, P) = (0.55, 0.06). Let ct be the reference set combined-test. Let D_{ct} be the perfect dictionary for the reference set.
  - We removed words not in ct from the dictionary D to increase precision. Sizes of the dictionaries decreased to 450K and 225K.
  - Created dic0294_p10 and dic0294_p20 (C= 0.55)
  - *Can you increase both precision and coverage?*

coverage and precision of improved dictionaries with respect to target sets

	Yahoo-test		Rockyou-test	
	Coverage	Precision	Coverage	Precision
dic0294	0.57	0.037	0.54	0.03
dic0294_c70	0.71	0.028	0.69	0.02
dic0294_c90	0.9	0.025	0.89	0.02
dic0294_p10	0.53	0.051	0.52	0.04
dic0294_p20	0.50	0.087	0.5	0.075

Actual cracking with improved coverage

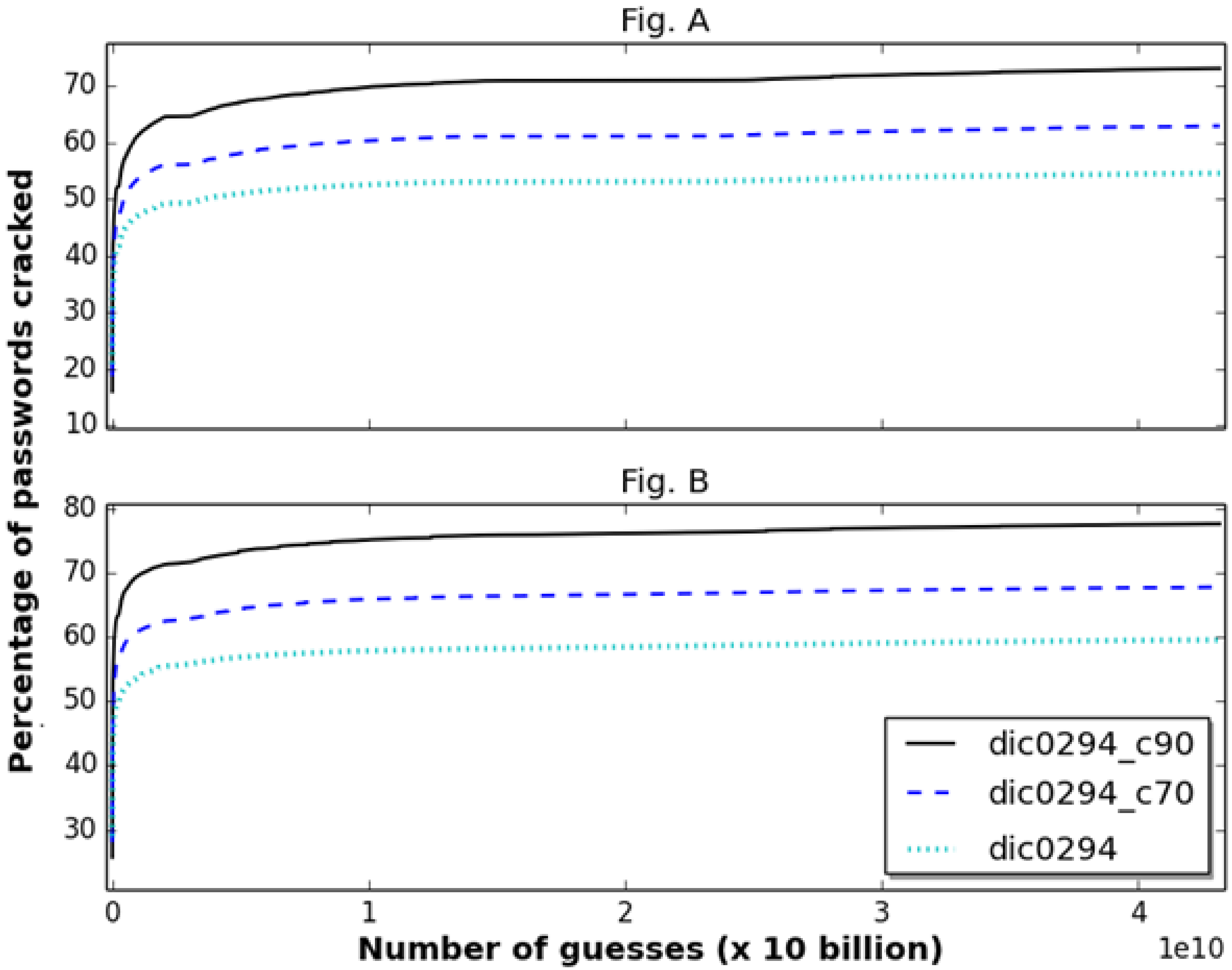


Fig A. Target is Yahoo-test

Fig B. Target is Rockyou-test

Actual cracking with improved precision

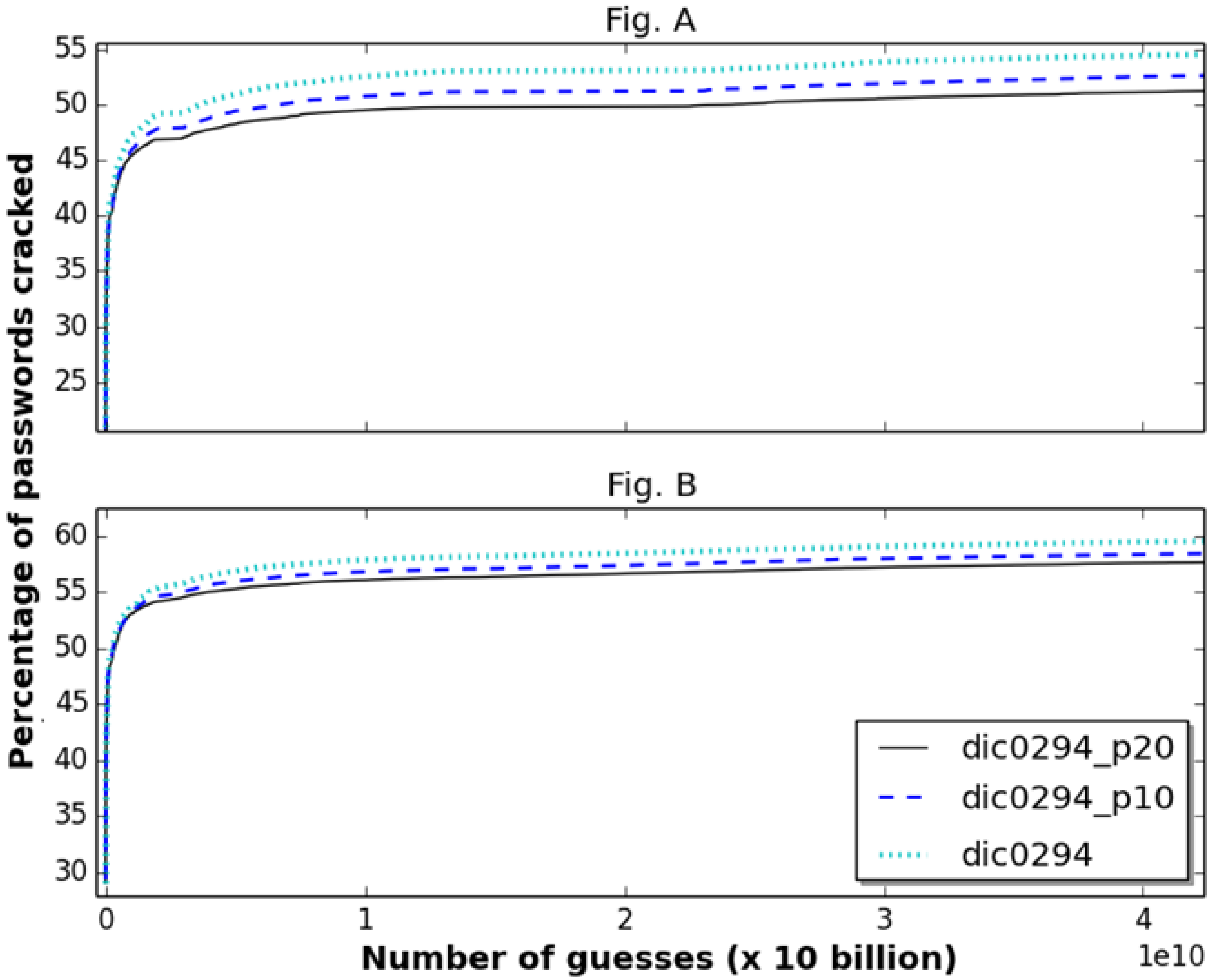


Fig A. Target is Yahoo-test

Fig B. Target is Rockyou-test

Dictionaries Summary

- Improving coverage and precision can be done
- Reference set idea seems good and may accurately reflect estimates of the utility of various dictionaries on target sets.
- Coverage seems more important than precision
- We were able to improve the cracking substantially by improving the dictionary.

Extensions

- Modeling Differences between Passwords
- Keyboard Combinations
- Better Identification of Alpha Strings
- Developing Better Attack Dictionaries
- LeetSpeak

Transformation of Words - LeetSpeak

Dictionary Word	Transformed Word
password	p@ssword
password	passw0rd
fool	F0ol
will	w1ll
facebook	faceb00k

How common are such replacements

Length	#non-leet	#leet	probability of LeetSpeak
4	1520	1	0.0006574621959237344
5	30657	40	0.0013030589308401473
6	129172	482	0.003717586807965817
7	89089	399	0.004458698372966208
8	79261	261	0.003282110610900128
9	44927	88	0.0019549039209152503
10	28317	35	0.0012344808126410836
11	14775	1	6.76773145641581e-05
12	8869	1	0.00011273957158962796
14	3301	1	0.0003028467595396729
16	1288	1	0.0007757951900698216

Defining replacement structure

Dictionary Word	Potential Replacement Structure
password	asso
leet	ee
sail	ail
bail	ail
fail	ail
randy	a
mars	as

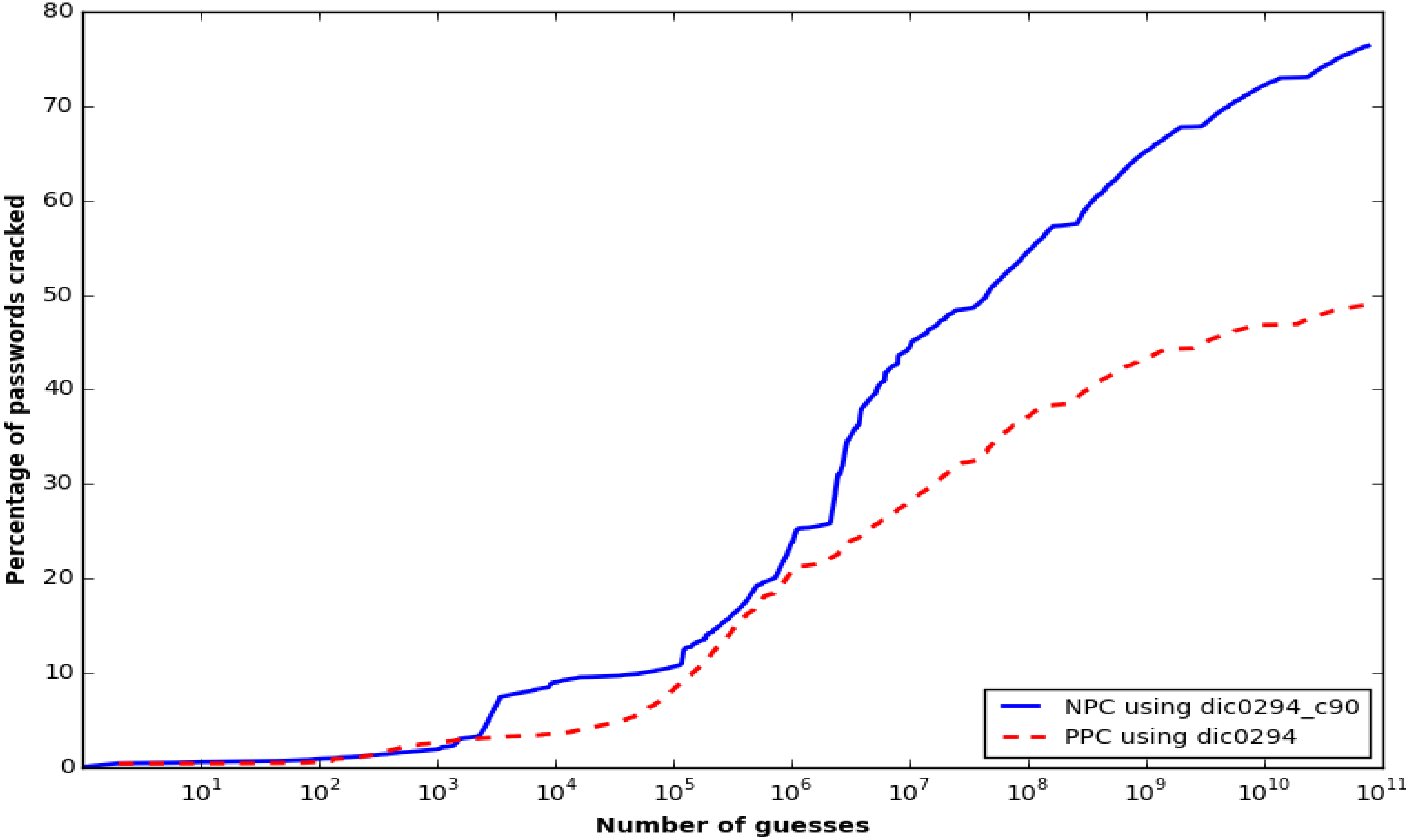
Specific Replacements

Potential Replacement Structure	Specific Replacement Structure	Probability
asso	SaNsNsSo	0.2156
asso	NaNsNsSo	0.7647
asso	NaSsSsSo	0.0196

Some Issues

- Multiple replacements for the same character
  - I and L can both be replaced by a “1”
- Is the password “111” a DDD or a EEE?
  - ILL may also be in the dictionary
- Whole word replacements or partial
- Smoothing

Results using all the techniques



Summary

- We have added many enhancements to make our approach much more effective and useful
- In particular, we have developed systematic approaches for keyboard combinations and identification of alpha strings
- We have defined a new approaches to modeling differences and targeted attacks
- We have explored the use of training dictionaries and attack dictionaries

Some references to our work

M. Dell'Amico, P. Michiardi and Y. Roudier. 2010. Password strength: an empirical analysis. *Proceedings of IEEE INFOCOM 2010*.

P. G. Kelley, S. Komanduri, M. L. Mazurek, R. Shay, T. Vidas, L. Bauer, N. Christin, L. F. Cranor, and J. Lopez. 2012. Guess again (and again and again): measuring password strength by simulating password-cracking algorithms. *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, pp 523-537.

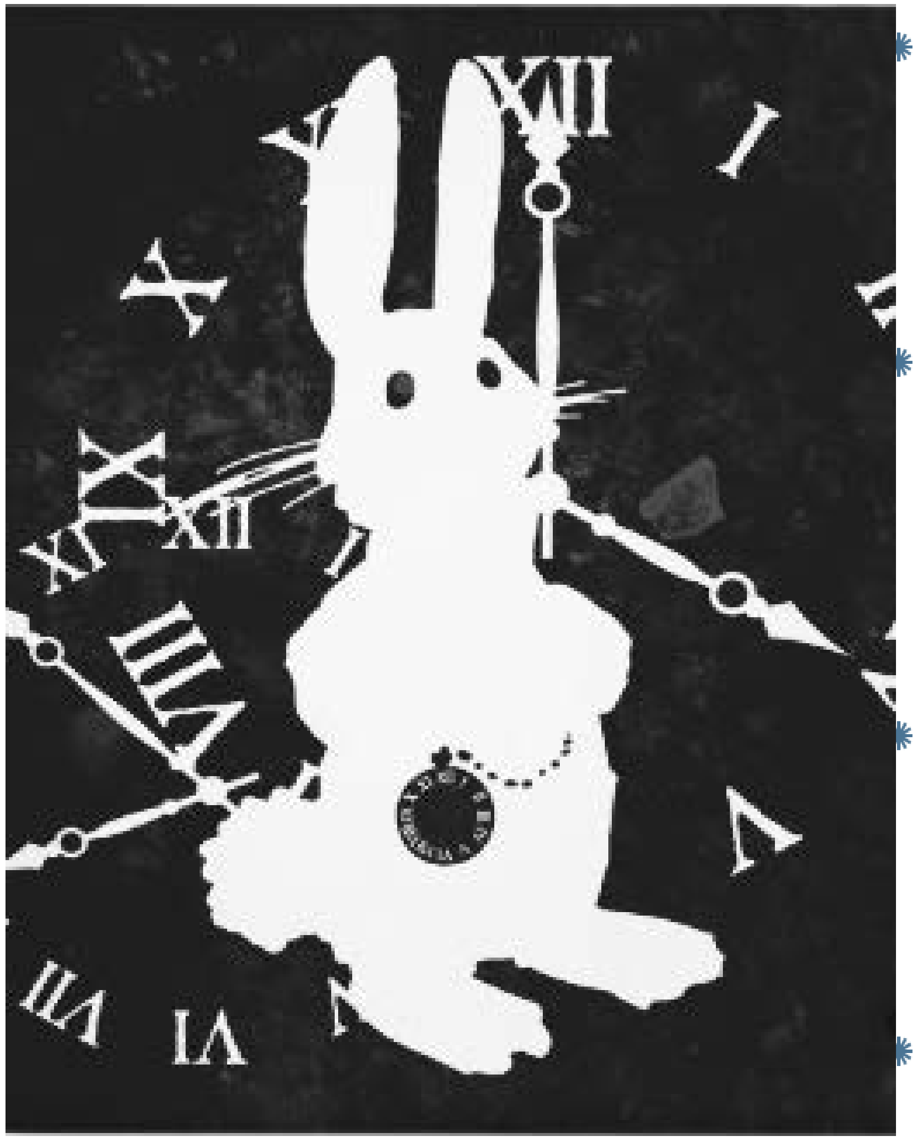
Y. Zhang, F. Monrose, and M. K. Reiter. 2010. The security of modern password expiration: an algorithmic framework and empirical analysis. *Proceedings of ACM CCS'10*.

Ur, Blase, Patrick Gage Kelley, Saranga Komanduri, Joel Lee, Michael Maass, Michelle L. Mazurek, Timothy Passaro et al. "How does your password measure up? The effect of strength meters on password creation." *In Proc. USENIX Security*. 2012.

Rao, Ashwini, Birendra Jha, and Gananand Kini. "Effect of grammar on security of long passwords." *Proceedings of the third ACM conference on Data and application security and privacy*. ACM, 2013.

Ari Juels and Ronald L. Rivest, "Honeywords: Making Password-Cracking Detectable," *preprint MIT CSAIL*, May 2, 2013. http://people.csail.mit.edu/rivest/pubs/JR13.pdf

Our work



- M. Weir, Sudhir Aggarwal, Breno de Medeiros, Bill Glodek, “Password cracking using probabilistic context free grammars,” *Proceedings of the 30th IEEE Symposium on Security and Privacy*, May 2009, pp. 391-405.

- M. Weir, S. Aggarwal, M. Collins, and H. Stern, “Testing metrics for password creation policies by attacking large sets of revealed passwords,” *Proceedings of the 17th ACM Conference on Computer and Communications Security (CCS '10)*, October 4-8, 2010, pp. 163-175.

- Shiva Houshmand, Sudhir Aggarwal, “Building better passwords using probabilistic techniques,” *Proceedings of the 28th Annual Computer Security Applications Conference (ACSAC ’12)*, December 2012, pp. 109-118.

- *Houshmand, S.; Aggarwal, S.; Flood, R., "Next Gen PCFG Password Cracking," Information Forensics and Security, IEEE Transactions on , vol.10, no.8, pp.1776,1791, Aug. 2015*

Thanks!
Questions/Comments?