Appendix A
Cleansing of our datasets

Contaminated datasets. Of particular interest is our observation that, there is a non-negligible overlap between the original Tianya dataset and 7k7k dataset. We were first puzzled by the fact that the password "`111222tianya`" originally lay in the top-10 most popular list of both datasets. We manually scrutinize the original datasets (i.e., before removing the email addresses and user names) and are surprised to find that there are around 3.91 million (actually 3.91*2 million due to a split representation of 7k7k accounts, as we will discuss later) joint accounts in both datasets. We realize that someone probably have copied these joint accounts from one dataset to the other.

Our cleansing approach. Now, a natural question arises: *From which dataset have these joint accounts been copied?* It is highly likely that these joint accounts were copied from Tianya to 7k7k, *mainly for two reasons*. Firstly, it is unreasonable for 0.34% users in 7k7k to insert the string "`tianya`" into their 7k7k passwords, while users from tianya.cn are natural to include the site name "`tianya`" into their passwords for convenience. The following second reason is quite subtle yet convincing. In the original Tianya dataset, we find that the joint accounts are of the form {user name, email address, password}, while in the original 7k7k dataset such joint accounts are divided into two parts: {user name, password} and {email address, password}. The password "`111222tianya`" occurs 64822 times in 7k7k and 48871 times in Tianya, and one gets that $64822/2 < 48871$. Therefore, it is more plausible for someone to copy *some* (i.e., 64822/2 of a total of 48871) accounts using "`111222tianya`" as the password from Tianya to 7k7k, rather than to copy all the accounts (i.e., 64822/2) using "`111222tianya`" as the password from 7k7k to Tianya and further reproduces $16460 (= 48871 - 64822/2)$ such accounts.

After removing 7.82 million joint accounts from 7k7k, we found that all of the passwords in the remaining 7k7k dataset occur even times (e.g., 2, 4 and 6). This is expected, for we observe that in 7k7k half of the accounts are of the form {user name, password}, while the rest are of the form {email address, password}, and it is likely that both forms are directly derived from the form {user name, email address, password}. For instance, both {wanglei, wanglei123} and {wanglei@gmail.com, wanglei123} are actually derived from the single account {wanglei, wanglei@gmail.com, wanglei123}. Consequently, we further divide 7k7k into two equal parts and discard one part. The detailed information on data cleansing is summarized in Table I of the main text.

Weaknesses in existing studies. In 2014, Li et al. [7] has also exploited the datasets Tianya and 7k7k. However, contrary to what we have done above, they think that the 3.91M joint accounts are copied from 7k7k to Tianya. Their main reason is that, when dividing these two datasets into the reused passwords group (i.e., the joint accounts) and the not-reused passwords group, they find that "the proportions of various compositions are similar between the reused passwords and the 7k7k's not-reused passwords, but different from Tianya's not-reused passwords". However, they have *never* explained what these "various compositions" are. Their explanation also cannot answer the critical question: why are there so many 7k7k users using "`111222tianya`" as their passwords? Hence, it would be more reasonable that they had removed 3.91*2 million joint accounts from 7k7k but not 3.91 million ones from Tianya. In addition, they did not observe the extremely *abnormal* fact that all the passwords in 7k7k occur even times. *Such contaminated data would highly lead to inaccurate results and unreliable comparisons*. For example, Li et al. [7] reported that there are 9,477,069 (30.67%) passwords in Tianya with consecutive exactly six digits, yet the actual value is 2.5 times larger: 23,358,248 (75.59%). For another example, Li et al. reported that there are 32.41% of passwords in 7k7k containing dates in "YYYMMDD", yet the actual value is 6 times lower: 5.42%.

We have reported this issue to the authors of [7], they responded to us and acknowledged this flaw in their journal version [6]. Unfortunately, Han et al. [6] still fail to clean the datasets properly in the journal version and address our revealed issue in an oversimplified (and crude) way: "we removed these duplicate passwords from both websites" [6]. As their journal version [6] is essentially a verbatim of [7], we mainly use [7] for comparison and discussion.

Appendix B
Detailed info about our 22 semantic dictionaries

We now detail how to construct our 22 semantic-based dictionaries, in order to make our work reproducible as well as to facilitate the community. "English_word_lower" is from http://bit.ly/2b2uPBX and it contains about 58,000 popular lower-case English words. "English_lastname" is a dictionary consisting of 18,839 last names with over 0.001% frequency in the US population during the 1990 census, according to US Census Bureau [3]. "English_firstname" contains 5,494 most common first names (1,219 male and 4,275 female names) in US [3]. "English_fullname" is a cartesian product of "English_firstname" and "English_lastname", consisting of 1.04 million most common English full names.

To get a Chinese full name dictionary, we employ the 20 million hotel reservations dataset [5] leaked in Dec. 2013. The Chinese family name dictionary includes 504 family names which are officially recognized in China. Since the first names of Chinese users are widely distributed and can be almost any combinations of Chinese words, we do not consider them in this work. As the names are originally in Chinese, we transfer them into Pinyin without tones by using a Python procedure from https://pypinyin.readthedocs.org/en/latest/ and remove the duplicates. We call these two dictionaries "Pinyin_fullname" and "Pinyin_familyname", respectively.

"Pinyin_word_lower" is a Chinese word dictionary known as "SogouLabDic.dic", and "Pinyin_place" is a Chinese place dictionary. Both of them are from [9] and also originally in Chinese, and we translate them into Pinyin in the same way as we tackle the name dictionaries. "Mobile_number" consists of all potential Chinese mobile numbers, which are 11-digit strings with the first seven digits conforming to predefined values and the last four digits being random. Since it is almost impossible to build such a dictionary on ourselves, we instead write a Python script and automatically test each 11-digit string against the mobile-number search engine http://ku.13131313131.com/.

As for the birthday dictionaries, we use date patterns to match digit strings that might be birthdays. For example, "YYYYMMDD" stands for a birthday pattern that the first four digits indicate years (from 1900 to 2014), the middle two represent months (from 01 to 12) and the last two denote dates (from 01 to 31). "PW with a l^+-letter substring" is a subset of the corresponding dataset and consists of all passwords that include a letter substring *no shorter than* l, and similarly for "PW with a l^+-digit substring".

APPENDIX C
A SUBTLETY ABOUT GOOD-TURING SMOOTHING ON PASSWORD CRACKING

There is a subtlety to be noted when implementing the Good-Turing (GT) smoothing technique. We denote f to be the frequency of an event, and N_f to be the frequency of frequency f. According to the basic GT smoothing formula, the probability of a string "$c_1 c_2 \cdots c_l$" in a Markov model of order n is denoted by

$$P(\text{“}c_1 c_2 \cdots c_{l-1} c_l\text{”}) = \prod_{i=1}^{l} P(\text{“}c_i | c_{i-n} c_{i-(n-1)} \cdots c_{i-1}\text{”}), \quad (1)$$

where the individual probabilities in the product are computed empirically by using the training sets. More specifically, each empirical probability is given by

$$P(\text{“}c_i | c_{i-n} \cdots c_{i-1}\text{”}) = \frac{S(count(c_{i-n} \cdots c_{i-1} c_i))}{\sum_{c \in \Sigma} S(count(c_{i-n} \cdots c_{i-1} c))}, \quad (2)$$

where the alphabet Σ includes 10 printable numbers on the keyboard plus one special end-symbol (i.e., c_E) that denotes the end of a password, and $S(\cdot)$ is defined as:

$$S(f) = (f+1)\frac{N_{f+1}}{N_f}. \quad (3)$$

It can be confirmed that this kind of smoothing works well when f is small, yet it fails for passwords with a high frequency because the estimates for $S(f)$ are not smooth. For instance, `12345` is the most common 5-character string in the Rockyou dataset and occurs $f = 490,044$ times. Since there is no 5-character string that occurs 490,045 times, N_{490045} will be zero, implying the basic GT estimator will give a probability 0 for $P(\text{“12345”})$. A similar problem regarding the smoothing of frequency of passwords has been identified in [2].

There have been various improvements suggested in linguistics to cope this problem, among which is Gale and Hill's "simple Good-Turing smoothing" [4]. This improvement is famous for its simplicity and accuracy. This improvement (denoted by SGT) takes two steps of smoothing. Firstly, SGT performs a smoothing for N_f:

$$SN(f) = \begin{cases} N(1) & \text{if } f = 1 \\ \dfrac{2N(f)}{f^+ - f^-} & \text{if } 1 < f < \max(f) \\ \dfrac{2N(f)}{2f - f^-} & \text{if } f = \max(f) \end{cases} \quad (4)$$

where f^+ and f^- stand for the next-largest and next-smallest values of f for which $N_f > 0$. Then, SGT performs a linear regression for all values SN_f and obtains a Zipf distribution: $Z(f) = C \cdot (f)^s$, where C and s are constants resulting from regression. Finally, SGT conducts a second smoothing by replacing the raw count N_f from Eq.3 with $Z(f)$:

$$S(f) = \begin{cases} (f+1)\dfrac{N_{f+1}}{N_f} & \text{if } 0 \leq f < f_0 \\ (f+1)\dfrac{Z(f+1)}{Z(f)} & \text{if } f_0 \leq f \end{cases} \quad (5)$$

where $t(f) = |(f+1) \cdot \frac{N_{f+1}}{N_f} - (f+1) \cdot \frac{Z(f+1)}{Z(f)}|$ and $f_0 = \min\left\{ f \in \mathbb{Z} \,\middle|\, N_f > 0, t(f) > 1.65\sqrt{(f+1)^2 \frac{N_{f+1}}{N_f^2}(1 + \frac{N_{f+1}}{N_f})} \right\}$.

In 2014, Ma et al. [8] introduced GT smoothing into Markov-based attacks to facilitate more accurate generation of password guesses, yet little attention has been paid to the unsoundness of GT for high frequency events as illustrated above. To the best of our knowledge, we for the first time well explicate the combination uses of GT and SGT in Markov-based password cracking.

REFERENCES

[1] J. Bonneau, "The science of guessing: Analyzing an anonymized corpus of 70 million passwords," in *IEEE S&P 2012*, pp. 538–552.

[2] ——, "Guessing human-chosen secrets," Ph.D. dissertation, University of Cambridge, 2012.

[3] R. A. Butler, *List of the Most Common Names in the U.S.*, Jan. 2016, http://names.mongabay.com/most_common_surnames.htm.

[4] W. Gale and G. Sampson, "Good-turing smoothing without tears," *Journal of Quantitative Linguistics*, vol. 2, no. 3, pp. 217–237, 1995.

[5] J. Goldman, *Chinese Hackers Publish 20 Million Hotel Reservations*, Dec. 2013, http://bit.ly/2aVKyBw.

[6] W. Han, Z. Li, L. Yuan, and W. Xu, "Regional patterns and vulnerability analysis of chinese web passwords," *IEEE Trans. Inform. Foren. Secur.*, vol. 11, no. 2, pp. 258–272, 2016.

[7] Z. Li, W. Han, and W. Xu, "A large-scale empirical analysis on chinese web passwords," in *Proc. USENIX SEC 2014*, pp. 559–574.

[8] J. Ma, W. Yang, M. Luo, and N. Li, "A study of probabilistic password models," in *Proc. IEEE S&P 2014*, pp. 689–704.

[9] *Sogou Internet thesaurus*, Sogou Labs, April 17 2016, http://www.sogou.com/labs/dl/w.html.