Of contraseñas, סיסמאות, and 密码

Character encoding issues for web passwords

Joseph Bonneau
Computer Laboratory
University of Cambridge
jcb82@cl.cam.ac.uk

Rubin Xu
Computer Laboratory
University of Cambridge
rx201@cl.cam.ac.uk

***Abstract*—Password authentication remains ubiquitous on the web, primarily because of its low cost and compatibility with any device which allows a user to input text. Yet text is not universal. Computers must use a character encoding system to convert human-comprehensible writing into bits. We examine for the first time the lingering effects of character encoding on the password ecosystem. We report a number of bugs at large websites which reveal that non-ASCII passwords are often poorly supported, even by websites otherwise correctly supporting the recommended Unicode/UTF-8 character encoding system. We also study user behaviour through several leaked data sets of passwords chosen by English, Chinese, Hebrew and Spanish speakers as case studies. Our findings suggest that most users still actively avoid using characters outside of the original ASCII character set even when allowed to. Coping strategies include transliterating non-ASCII passwords using ASCII, changing keyboard mappings to produce nonsense ASCII passwords, and using passwords consisting entirely of numbers or of a geometric pattern on the keyboard. These last two strategies may reduce resistance to guessing attacks for passwords chosen by non-English speakers.**

I. Introduction

Text passwords continue to dominate web authentication, with their universality and ease of deployment being cited as key factors behind their continued persistence [14], [19]. Yet the seemingly simple process of converting an abstract secret stored in human memory [15] into bits suitable for hashing and storage in a password scheme relies on several conversions which are not universal.

First, users often use a natural language to express a secret in the form of written text. Natural language is not universal, of course, and the choice of language impacts the authentication process. For example, a bilingual user might equally choose the strings `my_mom_loves_me` or `mi_mama_me_ama` to represent the same concept. Similarly, even a monolingual user might choose different linguistic expressions of the same concept, such as `my_mother_loves_me`, or an alternate representation of the same words of natural language such as `My-Mom-Loves-Me`. Computers still struggle to understand natural language, meaning that users must memorise a secret concept, the precise wording used to express the secret in some natural language and the exact textual representation of that wording. This process is prone to failure and usability studies suggest that a significant number of users will be unable to use a password they remember conceptually because they cannot reproduce the precise representation [33].

A further conversion must take place to convert the abstract concept of "text" into a sequence of bits suitable for computer manipulation. For example, the letter `m` at the beginning of the password above is commonly represented using the eight bits `01101101`. This process is known as *character encoding* and, despite decades of work towards a universal standard, there remain dozens of schemes in widespread use to map characters into sequences of bits.

In principle, users should be able to choose any character encoding scheme and servers can obliviously hash the bits with no concern about their higher-level meaning. In practice, servers often are not oblivious to higher-level characters and assume a specific encoding. The user's browser and operating system must also decide which encoding to transmit a password with, which is not always straightforward and can be handled differently by different browsers. As a result, several edge cases still exist which prevent the use of some natural-language characters at some websites.

Furthermore, the set of characters available may be limited by the user's system. Most computers, through both their operating system and physical keyboard, are optimised for inputting only a small set of characters. Entering other characters might require either typing complicated sequences of keys or selecting characters using a graphical interface in addition to the keyboard. For some languages, notably East Asian languages like Chinese, Japanese, and Korean, entering characters routinely requires a graphical interface to disambiguate the huge number of characters relative to the number of keyboard keys.[1] Graphical input methods are undesirable for passwords because they are slower and vulnerable to shoulder-surfing.

Thus, users face several obstacles when including char-

[1]There are alternatives to graphical entry. For example, the Wubi method for typing Chinese requires typing a sequence of characters which represent the individual strokes of a character. This method is much more complicated to learn, however, and is falling out of favour as graphical entry using language prediction has improved.

acters in their passwords outside the legacy ASII set of English-language letters, numerals and punctuation. Similar or even firmer restrictions exist for other elements of Internet infrastructure, notably uniform resource identifiers (URIs) [11], [12] and SMTP email addresses [20], [30], which have traditionally only allowed a subset of the ASCII characters to be used.[2] To our knowledge, this paper is the first attempt to examine character encoding problems for the Internet's password infrastructure. We'll give a brief summary of the technical aspects of character encoding on the web in Section II. In Section III we'll discuss restrictions placed by websites on password encoding and several bugs we identified. In Section IV we'll analyse user behavior using several data sets of leaked passwords.

II. Overview of character encoding

A. Brief history of character encoding

Encoding schemes to express characters numerically as *code points* existed for use in teleprinters for decades prior to the advent of the computer, dating at least to the original Morse code (amongst others) for telegraph communication which emerged in the 1840s.[3] Fischer provides a complete historical survey of character encoding [18]. Early computers used different encoding schemes adapted from teleprinters, leading to sufficient incompatibility that the ASCII (American Standard Code for Information Interchange) encoding scheme was proposed in 1960 [8] and standardised by 1963 [1]. ASCII encodes 95 printable characters consisting of 26 standard English letters and punctuation with 8 bits each, using only the lower 7 bits for encoding and reserving one bit for a parity check. This detail proved consequential for passwords, as the UNIX `crypt()` function for password hashing [26] compresses up to eight ASCII characters into a 56 bit DES key by discarding all of the parity bits.

ASCII was never intended as an international standard, being inadequate even for most languages using variants of the Latin alphabet. Various extensions were created over the decades following its introduction. The ISO 646 standard [2], published in 1973, specified some code points within ASCII as available for local use, enabling regional variants like the UK's BS 4730 [3] which replaces the # character at code point 35 in ASCII with the £ character.

The number of reserved characters in ASCII is too small to support most languages, leading to many 8-bit encodings which omit the parity bit of ASCII to support 128 extra characters. An influential example is the proprietary "code page 437" scheme of the original IBM PCs, which added many characters from Western European languages such as the Spanish ñ at code point 164. The ISO 8859 series of standards, first published in 1988, defined 16 standard extensions labeled 8859-1 through 8859-16. ISO 8859-1 [4], designed to encode Western European languages and often called Latin-1, has been the most widespread on the Internet.

Meanwhile, languages with more complex writing systems developed multi-byte encodings to support more than $2^8 = 256$ characters. In the 1980s, encodings were nationally standardised for Chinese, Japanese, and Korean [24]. Distinct standards emerged for Simplified Chinese in the People's Republic of China (GB2312 and its successor GBK) and Traditional Chinese in Taiwan, Singapore, and Hong Kong (Big5).

Thus while there were standardised encodings for most languages by the late 1980s they were generally incompatible with each other, motivating the proposal of a universal character set by Becker in 1988 [7] under the name "Unicode." The first version of the Unicode standard was published in 1991 [5]. The original version of Unicode, now obsolete, planned to support only 2^{16} code points and use a constant 16 bits per character.

Starting with Unicode version 2.0, more than 2^{16} code points are defined, leading Pike and Thompson to propose UTF-8 in 1993 [29]. UTF-8 is a variable-width encoding scheme which is backwards-compatible with ASCII. The first 128 code points (which map to identical characters by design in ASCII and Unicode) are encoded directly as a single byte, while higher code-points are represented by multiple bytes in a manner that allows for unambiguous decoding. UTF-8 is now generally recognised as the international standard encoding for all languages, with many standards bodies like the IETF mandating UTF-8 support [22].

B. Character encoding on the web

Unfortunately, the development of Unicode and UTF-8 was slightly too late for the early growth of the World Wide Web. In particular, the initial specification for HTML specified ISO 8859-1 as the character encoding for all HTML documents [9] and the initial HTTP specification designated ISO 8859-1 as the default encoding [10]. RFCs were quickly written for Internet support for languages like Hebrew [28], Chinese [32], Japanese [27] and Korean [16], none based on UTF-8. HTML specified a number of "character entity references" to specify non-ASCII characters using a sequence of ASCII characters, for example `&ntilde;` to indicate ñ. The standard also specifies "numeric entity references" to specify any other characters not encoded in ASCII. For example, `&#242;` and `&#x00F2;` are also valid representations of ñ based on its code point in Unicode (and also ISO 8859-1).

HTML documents may directly include characters using another encoding if this is declared, either with an HTTP header such as:

```
Content-Type: text/html;charset=utf-8;
```

or through an HTML tag in the document header such as:

[2]Upgraded standards exist for both URIs [21] and email addresses [31] to allow non-ASCII characters to be used, but neither is yet widely used.

[3]Even earlier than electric telegraphy, many character encoding systems were developed for use with optical semaphores.

```
<meta http-equiv="Content-Type"
content"text/html; charset=utf-8" />
```

Nevertheless, content has often been served without specifying a character encoding using either method, requiring web browsers to guess the encoding. Interpreting the encoding of a document of unknown origin has been the subject of academic research [23] and patents have been filed [25].

UTF-8 didn't surpass ASCII (commonly written as US-ASCII) as the most common encoding on the web until 2008 [17], though today it is estimated to be in use at 68.7% of websites [6]. ISO-8859-1 is still in use at 16.8% of websites, while US-ASCII has crashed to just 0.1%.

C. Password submission

Passwords on the web are typically submitted using an HTML `<input type="password">` form element. This hides the password from view as it is typed and disables graphical input mechanisms. Browsers otherwise handle the process of submitting a typed-in password to a server identical to any other text in an HTML form. This may require conversion if the user inputs characters in a different encoding than that used by the server. Forms may request submission using a particular encoding using the `accept-charset` attribute, though we haven't observed this for password entry, leaving browsers to rely on the character encoding they have used to render a form's parent HTML document.

Browsers cannot simply submit the encoded password directly, however, as the 1994 standard for Uniform Resource Identifiers (URIs) [12] requires text to be "percent encoded," in which any byte value outside a limited subset of ASCII must be converted to percent sign `%` followed by the byte value in hexadecimal. For example, to include an ñ in a URI, it can be represented as `%C3%B1` (using UTF-8) or `%F1` (using ISO 8859-1) amongst other possibilities. If the submit method of a form is `GET` or `POST` with `enctype="application/x-www-form-urlencoded"`, browsers must percent encode the password prior to submission. If the submit method is `POST` with `enctype="multipart/formdata"`, browsers can directly send the password encoded as raw bytes. This method is typically only used for forms which include a user-uploaded file but we did observe it used for several websites to receive password submissions.

The primary implication of this architecture is that passwords can be expanded into a large number of bytes per character as a result of being replaced with a numeric entity reference and being percent encoded. For example, the single Chinese character 爱, ('love') will be expanded as follows given different page encodings:

encoding	submission	length
GB2312	`%B0%AE`	6
UTF-8	`%E7%88%B1`	9
ISO 8859-1	`%26%2329233%3B`	14

If servers attempt to enforce length restrictions on password without distinguishing between bytes and characters this can lead to errors, as discussed in Section III-C.

We have also found edge cases of different behaviour between browsers. For example, in a page encoded using GB2312, there is no code point for the ñ character so the browser must replace it with an entity reference. In our testing, Internet Explorer chose to transmit `%26ntilde%3B` (`&ntilde;`) while Chrome and Firefox transmitted `%26%23241%3B` (`&#241;`). Neither choice is clearly endorsed by standards.

III. Server handling of passwords

In this section, we present findings from an informal survey of web sites' handling of non-ASCII passwords. We overview several bugs, some with security implications, resulting from servers not handling character encoding issues properly. Our goal was not to quantify the frequency of different server behaviour, but merely to identify unusual implementations and bugs at some popular websites, which was quite easy to do. We examined 24 sites, 12 primarily English-language and 12 primarily Chinese-language, with a mix of encodings at each site (about half UTF-8 and half ISO-8859-1 or GB2312, respectively). All of the flaws reported here were responsibly disclosed to the affected websites prior to publication.

A. Correctly supporting sites

In our testing, we were able to seamlessly use a password of over 100 non-ASCII characters at seven sites: Facebook, Twitter, Wikipedia, CSDN, Renren and Kaixin001(UTF-8), and DeviantArt (ISO-8859-1).

B. Policies against non-ASCII passwords

We found a total of 9 sites with an explicit policy barring non-ASCII characters in passwords: Google, Microsoft Live, Yahoo!, Amazon, Baidu, Taobao, Sina Weibo, Tianya and Youku. Most of these policies are implemented using client-side JavaScript which is possible to circumvent. We assume that few users would override this check and didn't test if further checks were made server side.[4] One Chinese-language site, Taobao, uses special ActiveX control in place of the normal password input to stop keylogging malware, with a side effect that passwords cannot be pasted in.

C. Broken password length policies

Two sites that we tested, IMDB and WalMart, incorrectly enforce their length requirements by counting bytes instead of characters. Both use the ISO-8859-1 encoding and apparently don't remove the percent encoding from submitted passwords. Thus, as described in Section II-C, individual UTF-8 characters can be escaped to as many as 14

[4]Servers should never assume client-side checking is successful, as this could lead to vulnerabilities if checks are overridden.

bytes each. IMDB enforces a maximum length of 64 bytes, restricting users to only 4 characters when not using ISO-8859-1. WalMart enforces a length of 6–16 bytes, meaning only a single non-ISO-8859-1 character can be used (or 2–5 non-ASCII ISO-8859-1 characters). This policy error both restricts users from using longer passwords and allows them to circumvent minimum length requirements easily.

D. Unicode code point truncation

Two top Chinese microblogging websites, Weibo and QQ, hash the user's password prior to transmission using JavaScript. Both sites forbid non-ASCII passwords, but don't impose any restrictions on submitted passwords during login, masking each character's code point returned by `CharCodeAt()` with `0xFF`. As a result, all of the Unicode characters a, Ł, c,[5] ∾, 屁 will be considered equivalent in a password because all have a code point $\equiv 41 \pmod{256}$. This means that different typed-in passwords will be mistakenly accepted. Since only ASCII passwords are accepted at either site, this is merely an oddity, though this could be a security issue if a similar implementation were used at other sites which allowed non-ASCII passwords.

E. DES-`crypt()` truncation

The popular Taiwanese gaming community site Gamer.com.tw limits password length to 4–8 UTF-8 characters. However, if the user enters a Chinese password such as 我的中文得很好, they will be able to log in successfully by sending only 我的中. The most likely explanation for the 8-character limit is that the server is using the obsolete DES-based `crypt()` to hash passwords. Because DES uses 56 bit keys, DES-`crypt()` discards all but the first 8 bytes of input, compressing them into a 56-bit key by truncating each byte to 7 bits. The high-order bits of each byte, originally reserved as a parity check in ASCII, are discarded.

At Gamer.com.tw, a user may register a Chinese password of up to 8 characters, such as 我的中文得很好, but will actually be able to log in successfully by sending only 我的中. The first three characters of this password encoded in UTF-8 take up 9 bytes by themselves, and DES-`crypt()` ignores the remaining bytes.[6] In fact the adversary only needs to get the first 2 bytes of the last character right, making the search even easier. In the worst case, the attacker may only need to guess two characters if the user choose characters which have a 4-byte UTF-8 encoding, which is true for many Chinese characters. This is a security flaw which might make guessing attacks much easier against users choosing non-ASCII passwords.

[5]In this example, the c is a Cyrillic letter, encoded distinctly from the visually similar Latin letter c.

[6]It appears that Gamer.com.tw correctly removes the percent-encoding prior to using `crypt()`. The bug would be even worse otherwise, potentially truncating all but the first character.

F. DES-`crypt()` string termination bug

Beyond the known limits on password length for DES-`crypt()` discussed above, we discovered a common implementation bug when processing non-ASCII characters: passwords are truncated upon seeing either of the bytes `0x00` or `0x80`, the latter of which can appear in the middle of non-ASCII passwords. For example, in UTF-8, the character À is represented by the two bytes `0xC380`, meaning that the passwords À and Àuseless will hash to the same value because anything after À will be ignored.

The bug is caused by the way DES-`crypt()` copies password to its internal buffer. Each byte is left-shifted by one position to discard the ASCII parity bit before copying to an internal buffer. Instead of checking for the termination character `0x00` in the original password though, the internal buffer is checked. As a result, any `0x80` byte terminates the loop prematurely. We found this buggy implementation in three widely-used software projects: FreeBSD's standard C library, PHP[7] and PostgresSQL. Though DES-based `crypt()` is no longer used to hash system passwords, our observation at Gamer.com.tw indicates it is still used at some websites and other application software.

G. Removal of non-ASCII characters from passwords

Two websites we tested (Gawker and Mop) accepted arbitrary UTF-8 passwords, but converted them at the server to another encoding (presumably ASCII) with inconvertible characters replaced with `?`. Gawker's problem lay in a faulty Java library to implement `bcrypt()` called `jcrypt()`. Mop.com has a similar problem that does not properly handle non-GBK characters. In either case, password guessing is made easier as an adversary can replace any potential characters outside the server's encoding with the default `?`, making passwords like $?^n$ an effective guess as they will match any password with n non-codeable characters.

IV. User choice of passwords

We now turn our attention to user choice of passwords, which we study by analysing data sets of passwords leaked from websites. We are limited to sets of cleartext passwords because using password cracking tools to invert hashed passwords would bias our observations of user choice.

A. Available data sets

We summarise the data sets we will study here. For websites which were compromised in the past, we rely on snapshots by the Internet Archive project to determine character encodings at the time of the compromise.

- rockyou.com: RockYou is a social application developer which develops games for Facebook, MySpace, and other online social networks. It was compromised

[7]PHP will use the system implementation of `crypt()` if available, but it maintains its own fork of the BSD implementation.

in December 2009 via SQL injection and 32 M passwords were leaked. The site appears to have always used UTF-8. At the time of the leak its home page was available in English, Spanish, Portuguese, Chinese, and Thai. No password requirements were in place at the time of the compromise.

- porn.com: Porn.com is a pornographic website which offers premium accounts. It was compromised in June 2011 as part of the "50 Days of Lulz" hacking incident, with about 25,000 usernames, emails, passwords, and names leaked. The site is available in English only and uses UTF-8. Passwords were required to be at least three characters long.
- www.nato.int/cps/en/natolive/e-bookshop.htm: The NATO e-Bookshop is a website for ordering and downloading official publications of NATO, the North Atlantic Treaty Organisation. It was compromised in June 2011 as part of the "50 Days of Lulz" hacking incident with about 10,000 usernames, emails, passwords, and names leaked. The site is available in English, French, Russian and Ukrainian and uses UTF-8. No password requirements were observed.
- wonder-tree.com: Wonder-Tree is a small religious meditation website. In May 2011 the site's administrators accidentally made a text file of about 1,000 usernames, emails, passwords, and postal addresses publicly accessible. The site is primarily in Hebrew with some English translation. Interestingly, the site's home page uses the Windows-1252 (Hebrew) encoding while the leaked data was entirely encoded in UTF-8.
- 70yx.com: 70yx is an online gaming website. A list of 10 million usernames and passwords were leaked as part of a major Chinese hacking incident in December 2011. 70yx is available in Chinese only and uses GB2312. It allows passwords of 6–16 characters from the ASCII subset only.
- csdn.net: CSDN is a Chinese-language forum site for software developers. A database of 6 million passwords were leaked in the same 2011 incident as 70yx, claimed to be a backup of the database of accounts from 2009. CSDN is available in Chinese only. It uses UTF-8 and imposes a 5-character minimum on passwords.

B. Types of characters chosen by users

In Table I, we provide an overview of the frequencies of different character classes within our available data. Note that all of our data sets were in effect encoded in UTF-8, with the one site not using UTF-8 (70yx) restricting users to ASCII characters for which the GB2312 encoding is identical.

1) Malformed passwords: The RockYou data set was the only one in which we observed passwords which were not well-formed UTF-8 strings. There were only 256 such passwords out of over 32M. It isn't possible to conclusively determine the encoding, particularly for short strings which may use encodings in which every string is valid. A manual inspection suggested that most of the non-UTF-8 passwords were ISO 8859-n variants, particularly ISO 8859-1. This might be due to non-compliant browsers submitting this encoding despite the character set specified by the page.

2) Non-ASCII passwords: The vast majority of the passwords observed in all data sets consisted only of characters in the traditional ASCII subset of UTF-8 (technically called the Basic Latin block). The Wonder-Tree data set contained about 2.5% passwords using characters outside of this range, in all but one case using the Hebrew block of Unicode.[8] Interestingly, 87.5% of the Wonder-Tree users did use Hebrew characters in their username, indicating that the majority of users actively decided not to use Hebrew characters in their passwords despite Hebrew being their preferred language.

All other data sets had fewer than 0.01% passwords containing non-ASCII characters. Still, the RockYou data set had 18,031 such passwords. Of these, 58.1% only included characters from the ISO-8859-1 (Latin-1) character set, mostly consisting of accented Latin letters. The remainder included a wide mixture, including Cyrillic, Greek, Hebrew, Arabic, Chinese, Japanese, and Korean, indicating that some tiny percentage of users do choose to use passwords in their native writing system.

Interestingly, in the CSDN data set only a handful of the non-ASCII passwords actually contained Chinese characters, with roughly ten times more passwords consisting exclusively of the black circle character ●, which may be an artifact of a buggy password manager or due to misguided copying of discreetly rendered passwords.

3) ASCII character preferences: Within the large majority of exclusively-ASCII passwords, what is most striking is a preference towards numeric-only passwords in the non-English language data sets, as seen in Table I. Fewer than 16% of users at RockYou only used digits in their passwords while 45–48% of users in the Chinese data sets did so. The Hebrew users at Wonder-Tree were in the middle with 38% of passwords being numeric-only. Similarly, 53–60% of users in the predominantly English data sets included at least one number in their passwords, compared to 65% of the Hebrew speakers and 87–90% of Chinese speakers.

A simple hypothesis to explain this phenomenon is that the Hindu-Arabic numeral system (the written digits `0`, `1`, …, `9`) is commonly used in both written Hebrew and Chinese,[9] making the digits more familiar for users with low fluency in English or other languages using the Latin alphabet. For Hebrew speakers, there is a second advantage

[8]The lone exception was a password which was entirely ASCII except for the special UTF-8 non-printing character which marks a change from left-to-right to right-to-left text rendering, which may have been included due to a copy-and-paste error.

[9]Both Hebrew and Chinese have separate numeral systems which are used for ceremonial or historical purposes, but Hindu-Arabic numerals are almost always used for practical applications.

language	site	size	digits 0–9 all	digits 0–9 some	letters a–z all	letters a–z some	letters A–Z all	letters A–Z some	alphanumeric all	alphanumeric some	adjacent keys
English	RockYou	32575653	15.9%	54.0%	41.7%	80.6%	1.5%	5.9%	96.3%	100.0%	3.1%
English	NATO Bookshop	11524	19.2%	53.4%	42.6%	78.6%	1.0%	10.3%	96.5%	100.0%	5.0%
English	porn.com	25934	27.7%	60.5%	35.9%	70.2%	0.9%	5.9%	97.8%	99.9%	8.4%
Hebrew	Wonder-Tree	1252	38.3%	65.3%	29.6%	54.6%	2.2%	5.7%	96.6%	98.1%	11.0%
Chinese	70yx	9072966	48.1%	90.8%	9.0%	50.7%	0.2%	1.0%	99.3%	100.0%	11.8%
Chinese	csdn	6428630	45.0%	87.1%	11.7%	51.4%	0.5%	4.6%	98.3%	100.0%	11.2%

Table I
CHARACTER CHOICES IN LEAKED PASSWORD DATA SETS

that numbers can be typed on a multi-input keyboard without switching the input mode from Hebrew to English. The observed preference for numeric passwords may explain earlier findings that many of the most common passwords on the web are used by all language groups [13].

Also of note, the rate of using non-alphanumeric ASCII characters, which include common English punctuation characters, tends to be higher for the English-language data sets than for the Chinese-language data sets (though it is very low overall). Again, this may be a factor of less familiarity with English/Latin punctuation amongst Chinese speakers of lower English fluency.

Finally, we performed a simple test for passwords which appear to be keyboard patterns such as `qwerty` or `1qaz2wsx`, listed as "adjacent keys" in Table I. For each password, we tested if transitions between consecutive characters (ignoring repeated characters) represented two keys which are adjacent on a standard US keyboard.[10] We considered a password to consist of adjacent keys if at least 75% of the transitions were adjacent on the keyboard, admitting a few false positives such as `sweety`. The rate of adjacent-key passwords was around 11% for the Chinese and Hebrew passwords and only 3.1% in the RockYou data set. Choosing a password as a geometric pattern may be another coping mechanism for dealing with less-familiar characters.

Comparing guessing statistics (which are computable for the larger distributions) indicates that the Chinese password datasets are weaker against guessing attacks. Using the metric $\tilde{G}_{0.5}$, which measures the expected amount of work required to break half of accounts in a guessing attack [13], the CSDN passwords had 20.3 bits of strength and the 70yx passwords only 15.7. For comparison, the RockYou passwords have 19.8 bits by this metric and a set of passwords collected from Yahoo! users 21.6 bits [13]. By the metric $\tilde{\lambda}_{10}$, which measures the efficiency of an attacker limited to just 10 guesses, the CSDN and 70yx passwords both had just 6.6 bits of strength, compared to 8.9 and 9.1 bits for the RockYou and Yahoo! passwords. An interpretation of this is that the tendency towards numeric passwords and keyboard patterns makes limited dictionary attacks easier, while it isn't clear what the impact is on more-exhaustive dictionary attacks.

[10]In mainland China, the US keyboard layout with no modifications is standard. A separate "Chinese (Taiwan)" keyboard is used in Taiwan, Hong Kong, and some other Chinese-speaking areas, but uses the standard US layout with additional Zhuyin character labels on most keys. Similarly, the standard Hebrew keyboard consists of the standard US keyboard with additional Hebrew labels on most keys.

C. Transliteration of passwords

We observed evidence in each data set of users actively changing a word in their native language into ASCII.

1) Chinese: The Pinyin transliteration system is a standard means of transliterating Chinese characters into the Latin alphabet. Because most Chinese speakers input Chinese characters by typing Pinyin onto a Latin keyboard and using recognition software, an obvious approach is for users to simply type the Pinyin representation of a Chinese word and use this as a password. We observe many examples of this among the most common passwords, for example "woaini" for 我爱你 (translating to "I love you") and "zhanghaomima" for 帐号密码 ("account & password").

We tested each password in the data set for correctness as a Pinyin word. Our method may have false positives in that some valid Pinyin sequences like `orange` (噢染个) represent a valid word in English (and many other languages). Still, we found that relatively few passwords were potentially valid Pinyin strings, with rates of 15.9% and 14.5% in the 70yx and CSDN data sets, respectively. Thus, even among the users not entering strictly numeric passwords, the majority of users do not appear to be entering Pinyin.

2) Hebrew: Unlike Chinese, most Hebrew speakers input Hebrew characters directly using individual key presses, with a standard Hebrew keyboard having 27 Hebrew letters available as well as 26 English letters in a dual mapping. Two users in the data set (out of just 28 using Hebrew characters), for example, chose the passwords **שדגכ** and **שדגכעי** which correspond to `asdf` and `asdfgh` on the Hebrew keyboard.

We manually found several examples of nonsense Latin passwords which, when run through a reverse English-Hebrew keyboard mapping, produced plausible Hebrew phrases. For example, the password `thigusnkcsu` corresponds to the Hebrew **אין עוד מלבדו** ("There is no one else but him"), which is a biblical quote from the Book

password	meaning	proper	transliterated	ratio
	ñ → n			
contraseña	password	408	218	34.8%
muñeca	doll	197	354	64.2%
cariño	affection, dear	104	153	59.5%
pequeña	little (girl)	87	72	45.2%
teextraño	I miss you	65	27	29.3%
	á → a			
teamomamá	I love you mom	2	151	98.7%
	ó → o			
código	code	5	110	95.7%
	ú → u			
música	music	2	1447	99.9%

Table II
TRANSLITERATION OF SPANISH PASSWORDS, ROCKYOU DATA SET

of Deuteronomy (4:35). Passwords like these demonstrate a coping strategy of typing a memorable Hebrew password with the keyboard mapping switched to English to ensure compatibility with legacy servers.

There are several closely-related standards for transliterating Hebrew phonetically into the Latin alphabet. Many examples exist of passwords which are the transliteration of a Hebrew word. For example, two users chose the password ahava, the transliteration of אהבה ("love"). Unlike the Pinyin system, reverse transliteration from arbitrary Latin characters to Hebrew is always possible, so there is no simple way to test the frequency of transliteration.

3) Spanish: While we don't have a data set specifically of Spanish-speaking users, we do find many likely Spanish passwords[11] within the RockYou data set. Spanish uses a variant of the Latin alphabet very similar to English, with the addition only[12] of the letter ñ (regarded as a letter proper and not an n with a diacritical mark) and the use of an acute accent over the five vowels á, é, í, ó and ú to indicate which syllable is stressed during pronunciation (unlike ñ these are not considered separate letters). To transliterate to the English alphabet, the stress accents are dropped and ñ is replaced with n.

To quantify the habits of Spanish speakers when choosing passwords, we identified a number of passwords in the RockYou data set which were obvious candidates for transliteration: passwords containing a non-ASCII character with a clear Spanish meaning, with both transliterated and original form occurring at least twice each, and with the transliterated version not being equivalent to any common English words which would make transliteration appear more common (ruling out cognates like américa and jesús).[13] We list some examples of the ratio of the proper version to the transliterated version in Table II.

We found very few good examples including stress accents, but those we did find were almost always transliterated-. We also observed dozens of relatively common transliterated passwords for which the proper veresion never appeared. For example, pajaro (bird) appeared 169 times while the proper pájaro was never observed. We might conclude that our examples are in fact outliers and the real rate of transliteration of stress accents is greater than 99% for Spanish passwords. This is consistent with the linguistic trend that Spanish speakers, particularly youth, frequently drop stress accents when writing emails or text messages, as meaning almost always remains clear.

In contrast, Spanish speakers seem relatively reluctant to transliterate ñ, with the most common passwords retaining the character a majority of the time.[14] The rate varied greatly between different passwords, variations which are strongly statistically significant as measured by Fisher's exact test ($p < 0.001$). It is difficult to estimate an aggregate transliteration rate for ñ, just as for the stress accents, because there are many linguistic collisions such as montaña ("mountain" in Spanish, chosen 3 times) and montana (a U.S. state, chosen 1,116 times) which bias the result.

The relative preference for retaining the ñ character may be due to the stronger linguistic effect of converting ñ → n, which markedly changes a word's pronunciation. A more plausible explanation is that Spanish keyboards always contain a key to directly type ñ, but usually require multiple keys to type a stress accent.

V. CONCLUDING REMARKS

We've sought to explore the effects of character encoding on passwords, a previously unexplored challenge of human-computer authentication. While our study is mostly exploratory and not a thorough quantitative analysis of all possible effects, evidence suggests both that sites still fail to support passwords using characters outside of those standard in the English language, and that users rarely attempt to do so. Authentication is a two-sided market and it is likely that these two effects have formed a positive feedback loop. While character encoding is finally beginning to converge on a universal standard in UTF-8 and many sites now support passwords in any script, it may take a long time to undo decades of conditioning to avoid non-ASCII passwords.

Beyond the inherent cultural bias this entails, speakers of non Latin-based languages appear much more likely

[11] We say "likely Spanish" because there are several languages closely related to Castilian Spanish including Catalan and Galician which share many common words. Indeed, even the term "Spanish" (instead of "Castilian") can be politically controversial due to the complex historical relationship between these languages. Spanish also shares many words with other common Romance languages like Portuguese, Italian, Romanian and French, as well as many English loanwords.

[12] Some authorities consider ch as a single letter in Spanish, though as of 2010 the Real Academia Española no longer endorses this and ch has always been encoded as a c followed by an h and not as a ligature.

[13] Any apparently transliterated word might actually be a word in another language—for example, pequena is a valid word in Portuguese (meaning "small") and carino in Italian (meaning "cute"). We limit ourselves to ruling out valid words in English, the dominant language of the data set.

[14] We observed negligible numbers of users potentially transliterating ñ → ny. It's harder to conclude this is a transliteration, as contrasenya is the correct Catalan word for "password".

to rely on numerals or keyboard patterns when choosing passwords. This may weakens passwords against guessing attacks, particular opportunistic online attacks attempting a small number of common passwords. It also makes guessing attacks universal, as an attacker can use numeric passwords to avoid creating language-specific dictionaries for different groups of users. Future work on password schemes should keep internationalisation in mind and remember that ASCII can only fully express the native language of a small minority of the world's population.

ACKNOWLEDGEMENTS

The authors thank Noam Szpiro for help interpreting Hebrew passwords and Claudia Diaz and Elsa Monica Treviño Ramírez for help with Spanish passwords. Joseph Bonneau is supported by the Gates Cambridge Trust.

REFERENCES

[1] American Standard Code for Information Interchange, June 1963. American Standards Association ASA X3.4-1963.

[2] 7-bit coded character set for information processing interchange. International Organisation for Standardisation, 1973. International Standard ISO 646.

[3] Specification for UK 7-bit coded character set. British Standards Institute BS 4730:1974, 1974.

[4] Information technology — 8-bit single-byte coded graphic character sets — Part 1: Latin alphabet No. 1. International Organisation for Standardisation, 1988. International Standard ISO 8859-1.

[5] *The Unicode Standard*, volume 1.0. Addison-Wesley, Reading, MA, 1991.

[6] Usage of character encodings for websites. W3Techs Web Technology Surveys, March 2012.

[7] J. D. Becker. Unicode 88. August 1988.

[8] R. W. Bemer. A proposal for character code compatability. *Communications of the ACM*, 3:71–72, February 1960.

[9] T. Berners-Lee and D. Connolly. Hypertext Markup Language - 2.0. IETF RFC 1866, 1995.

[10] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol — HTTP/1.0. IETF RFC 1945, 1996.

[11] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. IETF RFC 3986, January 2005.

[12] T. Berners-Lee, L. Masinter, and M. McCahill. Uniform Resource Locators (URL). IETF RFC 1738, 1994.

[13] J. Bonneau. The science of guessing: analyzing an anonymized corpus of 70 million passwords. In *SP '12: Proceedings of the 2012 IEEE Symposium on Security and Privacy*, 2012.

[14] J. Bonneau, C. Herley, P. C. van Oorschot, and F. Stajano. The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes. In *SP '12: Proceedings of the 2012 IEEE Symposium on Security and Privacy*, 2012.

[15] A. S. Brown, E. Bracken, S. Zoccoli, and K. Douglas. Generating and remembering passwords. *Applied Cognitive Psychology*, 18(6):641–651, 2004.

[16] U. Choi, K. Chon, and H. Park. Korean Character Encoding for Internet Messages. IETF RFC 1557, 1993.

[17] M. Davis. Moving to Unicode 5.1. *Google Blog*, May 2008.

[18] E. N. Fischer. The Evolution of Character Codes, 1874-1968. www.pobox.com/~enf/ascii/ascii.pdf, Acessed 2012.

[19] C. Herley and P. C. van Oorschot. A Research Agenda Acknowledging the Persistence of Passwords. *IEEE Security and Privacy Magazine*, 2012.

[20] J. Klensin. Simple Mail Transfer Protocol. IETF RFC 5321, October 2008.

[21] J. Klensin. Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework. IETF RFC 5980, August 2010.

[22] M. Kuhn. UTF-8 and Unicode FAQ for Unix/Linux. http://www.cl.cam.ac.uk/~mgk25/unicode.html, 1999.

[23] S. Li and K. Momoi. A composite approach to language/encoding detection. *19th International Unicode Conference*, 2001.

[24] K. Lunde. *CJKV Information Processing*. O'Reilly, 1st edition, 1999.

[25] D. Marple. System and method for determining a character encoding scheme. US Patent No 6701320, 2002.

[26] R. Morris and K. Thompson. Password Security: A Case History. *Communications of the ACM*, 22(11):594–597, 1979.

[27] J. Murai, M. Crispin, and E. van der Poel. Japanese Character Encoding for Internet Messages. IETF RFC 1468, 1993.

[28] H. Nussbacher and Y. Bourvine. Hebrew Character Encoding for Internet Messages. IETF RFC 1555, 1993.

[29] R. Pike and K. Thompson. Hello World. *USENIX Winter 1993 Conference Proceedings*, 1993.

[30] J. B. Postel. Simlpe Mail Transfer Protocol. IETF RFC 821, 1982.

[31] J. Yao and W. Mao. SMTP Extension for Internationalized Email. IETF RFC 6531, February 2012.

[32] H. Zhu, D. Hu, Z. Wang, T. Kao, W. Chang, and M. Crispin. Chinese Character Encoding for Internet Messages. IETF RFC 1922, 1996.

[33] M. Zviran and W. J. Haga. A Comparison of Password Techniques for Multilevel Authentication Mechanisms. *Computer Journal*, 36(3):227–237, 1993.