REGAP: A Tool for Unicode-Based Web Identity Fraud Detection

Anthony Y. Fu, Xiaotie Deng, and Liu Wenyin
Department of Computer Science, City University of Hong Kong, 83, Tat Chee Avenue, Kow Long Tong, Hong Kong SAR, P. R. China

ABSTRACT We anticipate the widespread usage of an internationalized resource identifier (IRI) or internationalized domain name (IDN) on the web as complement to universal resource identifier (URI). IRI/IDN is composed of characters in a subset of Unicode, such that a Unicode attack to IRI/IDN could happen. Hence, visually or semantically, certain phishing IRI/IDNs may show high similarity to the real ones. The potential phishing attacks based on this strategy are very likely to happen in the near future with the boosting utilization of IRI/IDN. We invented a method to detect such phishing attack. We constructed a unicode character similarity list (UC-SimList) based on char-char visual and semantic similarities and use a nondeterministic finite automaton (NFA) to identify the potential IRI/IDN-based phishing patterns. We implemented a phishing IRI/IDN pattern generation tool, REGAP, by which phishing IRI/IDN patterns can be generated into regular expressions (RE) for phishing IRI/IDN detection. We also address how such a tool can be applied to investigations.

KEYWORDS phishing, internationalized resource identifier (IRI), internationalized domain name (IDN), nondeterministic finite automaton (NFA), regular expression (RE), unicode attack, homograph attack, REGAP

INTRODUCTION

Phishing web pages are forged to mimic the legitimate ones to spoof users into leaking their security sensitive information. Phishers usually use visually and/or semantically similar user interfaces to spoof users. The user interfaces include web links in address bars and web pages. Unwary Internet users could be induced by the phishing web pages to expose their bank accounts, passwords, credit card numbers, etc.

Phishing web pages have recently been found to be increasing at a faster pace. More attention has been paid by the industry, academia, news media, as well as legislative entities. There were 20,209 phishing attacks reported to the Anti-Phishing Working Group (APWG) in May 2006 (according to the latest report from APWG). While the users have been gradually educated to such scams, phishers are resorting to more and more sophisticated tricks to avoid detection and people’s suspicion.
In this article, we review the Unicode-based web identity fraud (Unicode attack to IRI/IDN) and propose a counter measure to detect it. We first generate the similarity list (UC-SimList), which contains each pair of characters’ visual similarity. We investigate the semantic relationships of characters in a universal character set (UCS) and create semantic similarity list of UCS (UC-SimList). We use the two lists to generate visual/semantic similarity list of UCS (UC-SimList). The UC-SimList can be used to find similar characters of a given character.

We used NFA as the tool to construct phishing IRI/IDN patterns. We constructed NFA with the keywords of IRI/IDNs that we want to protect. If an IRI/IDN is suspected, we use the NFA to check whether the IRI/IDN is phishing. We used a regular expression to represent potential phishing IRI/IDN patterns. We implemented a phishing IRI/IDN pattern generator (REGAP) to generate the regular expressions of phishing IRI/IDN patterns and proposed a framework to build anti-phishing systems with REGAP. We also address the way to apply such a technique to the investigation.

The rest of this article is organized as follows. The following introduces related work. Next, we present a Unicode attack against the IRI/IDN and discuss Unicode character similarity and the generation, NFA-based phishing IRI/IDN pattern construction, and the phishing pattern generator, REGAP. Then we address the application to investigation. Finally, we conclude our work and discuss future work. To facilitate the deeper understanding of the article, we address the technical detail in appendixes A to C. We also introduce the usage of REGAP 0.30 in Appendix D.

**RELATED WORK**

A uniform resource identifier (URI) is composed of a sequence of characters from a subset of ASCII. Hence, URI is Latin language oriented, and it is used to represent Latin character–based semantics, such that URI provides a smooth way for people who are familiar with ASCII characters to understand and identify web resources.

Non-Latin language scripts usually have a direct mapping from their characters to A–Z through phonetic translations, such as Chinese Pinyin and Japanese Romaji. Therefore, it is not a big problem for URI to represent non-Latin languages. However, most people require being oriented and are used to localized operating systems, applications, and Internet services that can provide a natural way of human computer interaction. In fact most users are eager to use information systems in their language scripts. As a result, more protocols, standards, and systems are required to use a wider range of character sets. IRI/IDN is a complementary standard to URI. An IRI/IDN is a sequence of characters from a subset of UCS.

The most popular version of UCS (Ver. 2.1) uses 16 bits to represent a character versus 8 bits for ASCII, and it covers characters that represent most of the major languages’ scripts. IRI/IDN is more convenient for non-English speakers since it provides a more natural way for people who know Chinese but not English to input 中国银行,公司 rather than www.bank-of-china.com and for people who know Japanese but not English to input 東京三菱銀行,会社 rather than www.bank-of-tokyo-mitsubishi.com. Although Unicode has advantages of character representation, to our knowledge, there is no true-type font that can cover all characters in the latest version of UCS (Ver. 4.3). The most complete font that we can find is Arial Unicode Font MS, which is a complete Unicode font of Unicode Standard 2.1. It contains all the characters in Arial plus full fonts for Japanese, Chinese, Korean, Arabic, Hebrew, and different symbol characters and character ranges.

With the emergence of phishing scams in recent years, phishers have used a number of spoofing techniques to make the appearance of web links and web pages visually and semantically similar to the real ones. Many anti-phishing methods have been proposed. Previous research provides various duplicate document detection approaches that focus on plain text documents and use pure text features in similarity measure. The most effective strategy for detecting phishing is probably an active approach based on visual comparison of DOM generated from HTML, such as the one proposed by Liu et al. that uses the region-based approach to visual similarity of web pages. Another visual-based phishing detections technique is proposed by Fu et al., which is a phishing detection approach based on image level visual similarity assessment using the EMD algorithm. People also proposed anti-phishing methodologies from the cryptography perspective. Strong authentication of web pages is widely used in security-oriented websites. Legislative
action against Internet fraud is also being introduced in different countries.

The threat of Unicode attack and its methodology of counter measure are addressed in the literature.25,26 Following these research results, in this article, we review this attack to IRI/IDN and propose to use a regular generator to generate phishing patterns from word level (semantically) to character level (visually and semantically) automatically.

**UNICODE ATTACK TO IRI/IDN**

Recall that the IRN/IDN, a complement to the URI, is a sequence of characters of Unicode.12 The utilization of Unicode attack26 on an IRI/IDN that may be susceptible to a phishing attack could bring severe security problems since the universal character set (UCS)12 relies on a lot of visually and semantically similar characters. Figure 1 shows some of the similar characters for “3” and “bank.”

It is very easy for phishers to spoof users by replacing characters in a target IRI/IDN with similar ones. Although a phishing IRI/IDN looks very similar to or exactly the same to the targeted one, they are different at coding level. People could easily be deceived by this type of scam.

On the character level, Unicode attack is similar to homograph attack.27 However, on the word level, we would like to consider the following three as the most potential types of semantic obfuscation:

a. **Pronunciation-based replacement**: A phisher may replace some words of real IRI/IDN with other strings without changing the pronunciation. As shown in Figure 2, a phisher can change Chinese words to Pinyin, Japanese words to corresponding Romaji, and English to other commonly used forms.

b. **Abbreviation-based replacement**: A phisher may use the abbreviations of the keywords or acronyms of the real IRI/IDN for obfuscation. As shown in Figure 3, a phisher may change the keywords to their abbreviations, or conversely replace abbreviations into full names.

c. **Translation-based replacement**: A phisher may replace some words in a real IRI/IDN into the words of other languages for obfuscation. As shown in Figure 4, there could be many mutations of each part of an IRI/IDN through translation-based replacement.

**ANTI-PHISHING PATTERN GENERATION**

**Unicode Character Similarity**

We investigate the similarity of characters in UCS from two aspects: **visual similarity** and **semantic similarity**.

---

**FIGURE 1** Similar characters of “3” and “bank” in Arial Unicode MS font (adapted from UC-SimList28, and the code under each character are in hexadecimal form).

**FIGURE 2** Examples of IRI/IDN-based phishing obfuscation using the same pronunciations.
Visual similarity of two characters is measured by image similarity assessment algorithms. As our dataset is quite large, consisting of thousands of readable characters, we may prefer the simple, fast, but effective pixel-overlapping method, as discussed in appendix A. We denote the visual similarity list of Unicode characters as UC-SimListv. Although the construction of UC-SimListv is very important, it is also complicated. We have constructed the UC-SimList of three languages—English, Chinese, and Japanese—as discussed in appendix A. Finally, we merge UC-SimListv and UC-SimList to generate the UC-SimList, as also presented in appendix A. UC-SimList contains both visually and semantically similar characters of any given character in UCS.

Figure 5 shows the similar characters of Latin “A” and “a” in the UC-SimList, 0.8 (0.8 is the similarity threshold as defined in appendix A). In UC-SimListv, the Latin “A” and “a” are semantically similar. Hence the similar characters of “A” in UC-SimList 0.8 are as shown in Figure 6.

**Modeling the IRI/IDN-based Phishing Patterns with NFA**

We used NFA to model the potential IRI/IDN-based phishing obfuscation patterns. In general, the
modeling process is to convert the IRI/IDN list that we need to protect (e.g., banks’ domain names) into an NFA, which can be utilized to perform phishing IRI/IDN detection. It can be constructed by the following procedures: (a) we manually construct an NFA on the semantic level; (b) we replace each nonempty transition in the NFA with parallel-connected similar characters of this letter; (c) we replace each empty transition in the NFA with parallel-connected symbols that are valid in IRI. This final NFA is the one used to perform phishing IRI/IDN detection. The detailed process is available at appendix B.

Phishing IRI/IDN Pattern Generator

We built up a tool, regular expression generator for anti-phishing (REGAP ver. 0.30),\(^2\) to convert from the IRI/IDN NFA script directly to phishing detection regular expression. The IRI/IDN NFA script is an XML-style text string. It is a tree-like structure. For example, Figure 7 shows an IRI/IDN NFA script definition of “www.citibank.com”. In this script, we have one IRI (“www.citibank.com”), which includes two words (“citi” and “bank”). For each word, there are several representations. “Citi” can be represented as “citi,” “city,” “城,” “城市,” “花旗,” and “シティ.” “Bank” can be represented as “bank,” “银,” “行,” “銀行,” and “バンク.” The generated regular expression is shown in Figure 8. The delimiter-like character set, \(\Sigma\), is defined in Figure 9. \([0,2]\) indicates (a standard representation of regular expression) that the delimiter can repeat twice at the most at each position. We address the user interface and usage of REGAP in appendix D in detail. The regular expressions generated by REGAP are standard, such that it can be easily adapted to Perl, Java, C#, Python, PCRE, PHP, VI, JavaScript, Shell tools, etc. REGAP also uses the regular expression to detect phishing IRI/IDNs, as shown in appendix D.
APPLICATION TO INVESTIGATION

With the popularity of IRI/IDN, many web sites come to utilize this technology, especially in the countries that are not using Latin characters in their language scripts. This phenomenon opens a door for Unicode-based identity fraud. People can easily register any IRI/IDN from most of the DNS registrars. We would like to address three ways of application to investigations as follows.

Help Investigators to Detect Similar IRI/IDN

There are around 70 million domain names in the world. We take “www.citibank.com” as an example. Figure 10 shows the 100% visually/semantically similar characters to “www.citibank.com.” There are $4^w \times 4^w \times 9^c \times 6^t \times 14^i \times 5^b \times 7^a \times 5^n \times 5^k \times 9^c \times 7^o \times 5^m - 1 = 186,701,769,999$ possible faked IRI/IDN combinations in total. When we count in the obfuscations on the word level (as discussed previously) or reduce the similarity threshold of characters, there will be a lot more mutations of “www.citibank.com.” Phishing IRI/IDNs can be hidden anywhere. Hence, it is impossible for a human to manually detect similar IRI/IDN from such a large domain name pool. REGAP is a tool that can help a lot. Our experiments (in C# code) show that 70 million domain names can be scanned by the CitiBank pattern, as shown in Figure 8, in 80 minutes on a Pentium M 1.3-GHz 1GB RAM laptop.

<table>
<thead>
<tr>
<th>Latin Character</th>
<th>100% Visually/Semantically Similar Characters</th>
</tr>
</thead>
<tbody>
<tr>
<td>0077:w 0057:w</td>
<td>FF57:w FF37:w</td>
</tr>
<tr>
<td>0063:c 0043:C</td>
<td>0441:c FF43:c 217D:c 03F2:cFF23:C 216D:C 0421:C</td>
</tr>
<tr>
<td>0069:i 0049:I</td>
<td>0399:i 04C0:i 2170:i 0456:i FF49:i 0406:i FF29:i</td>
</tr>
<tr>
<td></td>
<td>217C:i FF4C:i 006C:i 01C0:i 2160:i</td>
</tr>
<tr>
<td>0074:t 0054:T</td>
<td>FF34:T FF54:t 0422:T 03A4:T</td>
</tr>
<tr>
<td>006E:n 004E:N</td>
<td>FF4E:n FF2E:N 039D:N</td>
</tr>
<tr>
<td>006B:k 004B:K</td>
<td>FF4B:k 212A:K FF2B:K</td>
</tr>
<tr>
<td>006F:o 004F:o</td>
<td>FF2F:o 039F:o 043E:o 03BF:o FF4F:o</td>
</tr>
<tr>
<td>006D:m 004D:M</td>
<td>039C:M FF4D:m 217F:m</td>
</tr>
</tbody>
</table>

FIGURE 9  Delimiter-like character set $\Sigma$.

FIGURE 10  One hundred percent visually/semantically similar characters to “www.citibank.com.”
For example, “www.citibank.com” is a security-sensitive web site, and we want to find all possible phishing IRI/IDNs from the domain name list. With the help of REGAP, we can generate a regular expression or represent the potential phishing IRI/IDN pattern, as shown in Figure 8. Figure 11 shows some examples of different types of obfuscation that can be recognized by this pattern.

Help Investigators to Collect Evidence through DNS Registrars

The application just presented is relatively passive, because we can only dig out the existing phishing IRI/IDNs. The phishing IRI/IDNs may have been there for quite a while. There are around 1 million domain names registered in one day.29 Hence, it could be a good approach to install the phishing IRI/IDN detection module to DNS servers and help monitor the domain name registration. We can create a list of the security-sensitive web sites’ patterns and use the phishing IRI/IDN detection module to detect and/or trace the potential attackers. Hence the attackers could be sued in real-time.

Our experiments (in C# code) show that a laptop with Pentium M 1.3-GHz and 1GB RAM can help register one new domain name while protecting 14,000 security-sensitive domain names in one second. Therefore, such an application performance is feasible from a technical aspect.

Help DNS Registrars to Provide a New Type of Service

To some extent, it should be reasonable to block the registration of potential phishing IRI/IDNs rather than to allow them to register and then catch them. REGAP can be very helpful to provide such service for DNS registrars that people are not allowed to register visually/semantically similar IRI/IDNs, and the performance is not a problem (similar to the performance just described).

However, in the real world, we may be concerned more about freedom of speech. People are not expecting such limitation to registering new domain names. Therefore, this kind of application is still conditional to some important aspects, such as the policy of the government, humanity, Internet security requirement, industrial promotion, etc.

CONCLUSION AND FUTURE WORK

In this article, we addressed the Unicode attack to IRI/IDN. We built up the UC-SimList, which can be used to find similar characters for a specified one easily. We proposed an effective detection approach to this problem using NFA and address its construction method. We built up a potential phishing IRI/IDN pattern generator REGAP Ver. 0.30, which can facilitate the generation of regular expression from the keyword-level NFA.

<table>
<thead>
<tr>
<th>Type of phishing IRI/IDN</th>
<th>Phishing IRI/IDN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character level visually similar</td>
<td>c i t i b a n k</td>
</tr>
<tr>
<td>Character level semantically similar</td>
<td>c l t l B A N K</td>
</tr>
<tr>
<td>Word level pronunciation</td>
<td>c i t y b a n k</td>
</tr>
<tr>
<td>Word level different language</td>
<td>花 旗 银 行</td>
</tr>
<tr>
<td>Word level abbreviation</td>
<td>花 银</td>
</tr>
<tr>
<td>Noise insertion</td>
<td>c i t i y - b a n k</td>
</tr>
</tbody>
</table>

FIGURE 11 Examples of phishing IRI/IDNs that can be detected (in these examples, the prefix ["www."]) and suffix [".com"] are omitted).
We used the simple pixel-overlapping method to measure visual similarity of characters. More accurate methods, like OCR, could definitely be employed. The current version of UC-SimLists only includes English, Chinese, and Japanese characters. We are expecting that more languages in UCS could be investigated in future work. We constructed the keyword-level NFA manually at this stage. Although the atomization of this process may involve more complicated semantic analysis, it is still possible to find an automatic way to do it, such as to construct a word–word similarity list and use it to make word replacement.

We discussed three possible approaches to help search phishing IRI/IDNs, collect phishing evidence, and/or provide a new domain name registration service. The anti-phishing framework we proposed previously could be easily adapted to various systems that can process the content of email, instant messages, BBS, chat rooms, files, etc., to perform phishing IRI/IDN searching and/or monitoring. It will be quite useful for evidence collection in investigations. Another good usage of REGAP is to use it as a tool to provide a special domain name registration service that registrars can help you protect all domain names that are acceptable to the potential phishing IRI/IDN patterns. However, it could cause some limitations to domain name registration.

All of our experiments on testing the performance are in C# code. Hence, the performance of any applications discussed should hopefully be much faster when we code in C. More performance evaluation should be carried out in future work.

Although the phishing IRI/IDN attack is not usual, we consider the countermeasure proposed in this article as a paradigm that can be used in the future. We also need to evaluate the false-positive detection when we can collect a number of real attack cases.

ACKNOWLEDGEMENTS

The work described in this article was fully supported by grants from the City University of Hong Kong (Project No. 7001842 and 7001975) and the China Semantic Grid Research Plan (National Grand Fundamental Research 973 Program, Project No. 2003CB317002). We thank Markus Jakobsson and Judie Mulholland for the useful discussion. We also thank the four anonymous referees for their constructive comments and suggestions.

NOTES

1. IRI is a generalization of the uniform resource identifier (URI), which is in turn a generalization of the uniform resource locator (URL). While URLs are limited to a subset of the ASCII character set, IRIs may contain characters from the universal character set (Unicode/ISO 10646). Basically, an IRI is the internationalized version of a URI.

2. IDN is an Internet domain name that (potentially) contains non-ASCII characters. Such domain names could contain letters with diacritics, as required by many European languages, or characters from non-Latin scripts such as Arabic or Chinese.

3. Unicode attack is caused by the coexistence of a large number of visual/semantically similar Unicode strings. On the character level, the visually similar Unicode attack is homograph attack.

4. NFA is a finite state machine where for each pair of state and input symbol there may be several possible next states. We can use it to recognize a string of a certain pattern. When the last input symbol is consumed the NFA accepts if and only if there is some set of transitions it could make that will take it to an accepting state. Equivalently, it rejects if no matter what choices it makes it would not end in an accepting state.


7. UCS is a character encoding defined in international standard ISO/IEC 10646. It contains nearly a hundred thousand abstract characters, each identified by an unambiguous name and an integer number called its code point.


10. “中国银行” is pronounced “Zhong Guo Yin Hang” and stands for “Bank of China”; “公司” is pronounced “Gong Si” and for “Company.”

11. “東京三菱銀行” is pronounced “Dou Kyou Mitsu Bishi Gin Ko” and stands for “Bank of Tokyo-Mitsubishi”; “会社” is pronounced “Kai Shya” and for “Company.”


18. DOM is a description of how an HTML or XML document is represented in a tree structure. DOM provides an object oriented application programming interface that allows parsing HTML or XML into a well defined tree structure and operating on its contents.


APPENDIX A. UC-SIMLIST

The visual similarity calculated with this method is empirically general for most fonts since characters have the same style in the same font, such that similar character pairs are still similar when we use another font to calculate the similarity of the pairs of characters. We used Unicode Arial MS Font Ver.1.0113 as our character image generation font since, among all fonts we could find, this font covers the largest number of characters. It covers most of the characters in the latest Unicode standard Ver. 4.312 and all characters of Unicode Ver. 2.1.12 We calculate the similarity of each pair of characters using the pixel overlapping percentage of them. For example, when we want to calculate the similarity of two characters, we one character onto the other and align them and count the number of the mutual pixels in the two glyphs; the similarity value is the mutual pixel number divided by the larger glyph’s pixel number. Our experiments show that when we set the size of Arial Unicode Font to 30 points, it performs well to calculate the similarities of character pairs without losing accuracy. It takes about one minute to compute the similarities of one character to all the others. Hence, it took 29 days to generate the entire UC-SimList, on an ordinary PC (Pentium IV 2.4-GHz single CPU and 512-MB RAM). We selected a threshold to define whether a pair of characters is similar. We also found from experiments that 0.8 is a good choice to simulate the human eye’s classification curve. UC-SimList, with different thresholds are available.28

The current version of UC-SimList includes English, Chinese, and Japanese and is available online.28

We leave other language parts as an open engineering problem and expect people could add and more languages to UC-SimList as future work.

APPENDIX B. THE THREE STEPS TO CONSTRUCT IRI/IDN NFA

Construct an NFA on the Semantic Level

Suppose we have a list of IRIs $L_{IRI}$ that need to be protected. We construct the keyword-level pattern NFA $NFA_{kw}$ from $L_{IRI}$; i.e., $L_{IRI} \rightarrow NFA_{kw}$. We find the keywords for each IRI, expand the keywords semantically, connect them properly with empty transitions to form an isolated NFA, and then merge the isolated NFAs into one NFA $NFA_{kw}$ in a parallel way.

Suppose we have an IRI $i \in L_{IRI}$. We convert $i$ into sequences of separated words using the three most potential types of semantic expansion of keyword in the Unicode Attack to IRI/IDN section to construct $NFA_{kw}$. We denote the keyword-level pattern NFA as:

$$NFA_{kw} = \bigcup_{i \in L_{IRI}} NFA_{kw}^i$$

where $NFA_{kw}^i = \bigcup_{1 \leq m \leq N_i} NFA_{m}$ is the keyword-level pattern NFA of $i$. $N_i$ is the number of possible sequential combinations of the expanded keywords of $i$, $\subseteq$ denotes one such combination, where $1 \leq m \leq N_i$ and “$\cup$” denotes parallel connection relationship of NFAs. We construct $NFA_{kw}$ manually because the number of IRIs that need to be protected is not big and we can construct $NFA_{kw}$ for each $i \in L_{IRI}$ carefully. In a practical system, we use XML to represent $NFA_{kw}$, as is discussed in previously. The $NFA_{kw}$ is generated by merging all $NFA_{kw}^i, i \in L_{IRI}$. The merging process can be accomplished by merging all initial state into a unique $q_0$, and final states into a unique $\Omega$. We construct the character-level pattern NFA $NFA$ from $NFA_{kw}$; i.e., $NFA_{kw} \rightarrow NFA$. This process can be accomplished as in the following two sections.

Replace the Nonempty Transitions

Replace nonempty characters in $NFA_{kw}$ with parallel-connected corresponding similar characters that can be found from UC-SimList by empty strings, and we denote it as $NFA_i$; i.e., $NFA_{kw} \rightarrow$
We get the similar characters of each character of \{c, i, t, y, b, a, n, k, \} from UC-SimList 0.8. Figure B1 shows the similar character sets of these characters; each element is represented by a character followed by its hexadecimal code. We denote the NFA formed by similar character set of characters. Figure B2 shows the examples of \(S(c), S(b), S(\text{银})\). The similar characters in each set are connected in a parallel way. We replace each character \(\text{char} \in \text{NFA}_{kw}^{c}\) with \(S(\text{char})\) to generate \(\text{NFA}^{c}\) as shown in Figure B2. There could be more IRIs other than \(i=\text{"www.citybank.com"}\) in \(L_{IRI}\). They are represented in a simplified way at the upper and lower position of \(\text{NFA}_{kw}^{c}\), where we denote the NFA generated from other IRIs with \(\text{NFA}^{c}\).

### Replace the Empty Transitions

We replace each empty transition \(\lambda\) in \(\text{NFA}^{c}\) with a transition with all delimiter characters in a parallel way and call it a delimiter transition (DT). The delimiter-like character set is denoted as \(\Sigma\), which includes the empty string \(\lambda\) and all characters look like the 22 valid symbols (’-’, ’.’, ’_’, ’~’, ’:’, ’/’, ’?’, ’#’, ’[’, ’]’, ’@’, ’!', ’$, ’&’, ’’’, ’(',')’, ’*, '+', ’,’’, ’;’, '='). We denoted the generated NFA as \(\text{NFA}^{c}\), i.e., \(\text{NFA}^{c} \rightarrow \text{NFA}^{c}\) as shown in Figure B2. More concatenated DTs can be used to replace the empty string in \(\text{NFA}^{c}\) and each DT means one appearance of delimiter appearance in the context. We used one DT in our approach for simplicity. So far, we

<table>
<thead>
<tr>
<th>Querying Character</th>
<th>Form1</th>
<th>Form2</th>
</tr>
</thead>
<tbody>
<tr>
<td>c:0063</td>
<td>c:0063</td>
<td>c:02F2</td>
</tr>
<tr>
<td>c:0441</td>
<td>c:0441</td>
<td>c:0460</td>
</tr>
<tr>
<td>c:0454</td>
<td>c:0454</td>
<td>C:00E7</td>
</tr>
<tr>
<td>c:0254</td>
<td>c:0254</td>
<td>O:FF2F</td>
</tr>
<tr>
<td>i:0069</td>
<td>i:0069</td>
<td>i:2170</td>
</tr>
<tr>
<td>i:0056</td>
<td>i:0056</td>
<td>i:00A1</td>
</tr>
<tr>
<td>i:03AF</td>
<td>i:03AF</td>
<td>i:FF51</td>
</tr>
<tr>
<td>i:006C</td>
<td>i:006C</td>
<td>i:01C0</td>
</tr>
<tr>
<td>i:0049</td>
<td>i:0049</td>
<td>i:FF4C</td>
</tr>
<tr>
<td>i:04C0</td>
<td>i:04C0</td>
<td>i:1E8B</td>
</tr>
<tr>
<td>i:0399</td>
<td>i:0399</td>
<td>i:217C</td>
</tr>
<tr>
<td>i:00ED</td>
<td>i:00ED</td>
<td>i:FF01</td>
</tr>
<tr>
<td>i:01C3</td>
<td>i:01C3</td>
<td>i:00ED</td>
</tr>
<tr>
<td>t:0074</td>
<td>t:0074</td>
<td>t:FF54</td>
</tr>
<tr>
<td>t:0167</td>
<td>t:0167</td>
<td>t:01AD</td>
</tr>
<tr>
<td>t:1E6F</td>
<td>t:1E6F</td>
<td>i:01AB</td>
</tr>
<tr>
<td>f:0006</td>
<td>f:0006</td>
<td>T:0186</td>
</tr>
<tr>
<td>y:1E5F</td>
<td>y:1E5F</td>
<td>y:01B4</td>
</tr>
</tbody>
</table>

**FIGURE B1** Similar character sets of each character in \{c, i, t, y, b, a, n, k, \} ("花, 行, 城, 市, 銀, 行, " denotes that Form 2 is the same as Form 1, and "市, 行, シ, " denotes that Form 2 is the omitted content).
have the key character pattern NFA of $L_{IRI}$; i.e., $NFA_c$, which is the final NFA that can be used to detect phishing IRI.

**APPENDIX C. GENERATE THE PHISHING IRI/IDN PATTERN IN REGULAR EXPRESSION**

Regular expression (RE) is equivalent to NFA. We convert $NFA_c$ to regular expression to ease the adaptation approach to anti-phishing systems. As $NFA_c$ is generated from $L_{IRI}$ with specified rules, the structure of $NFA_c$ is not complicated. We can apply the following procedure to convert $NFA_c$ to a regular expression. We use "|" to denote union, "" (empty string) for concatenation, "{m,n}" for repetition number between m and n. In our practical approach (See Conclusion and Future Work), we configure m to be 0 and n to be 1 for simplicity. Figure C1 shows the regular expression generation procedure in our approach.

Figure C2 shows the regular expression generated from $NFA_c$, where $i = \text{"www.citibank.com".}$. We represent the regular expression string generated in step c as $\Gamma (\text{char})$, where char is the same character in $S (\text{char})$. Figure C3 shows an example of $\Gamma (\text{char})$ when char = “c” where we use “u” + Unicode to represent a character. The regular expression of $NFA_c$ is the concatenation of all regular expressions of $NFA_i (i \in L_{IRI})$ with “|”, as shown in step b of the regular expression generation procedure.

**APPENDIX D. HOW TO USE REGAP**

REGAP is a tool that can help generate regular expressions to detect phishing IRI/IDNs. The current version of REGAP is 0.30. We demonstrate the usage of REGAP in the following steps.
Step 1. We can follow the format as shown in Figure D1 to write the IRI/IDN NFA script. One script can include many IRI/IDNs. Each IRI/IDN has a name (between "<IRINAME>" and "</IRINAME>") and is composed of one or more semantic word(s). Each semantic word has a name (between "<WORDNAME>" and "</WORDNAME>") and is composed of one or more sub-word(s). Comments can be added anywhere except the area between any pair of "<SW>" and "</SW>" tags.

Step 2. When we finish step 1, we can generate the regular expression according to the given IRI/IDN NFA script. We go to the "Regular Expression" tab pad and click "Generate>>" button. The generated regular expression is shown in Figure D2. We can also control the generation precession using different configurations. The "Replace \( \Sigma \) with delimiters" checkbox controls the representation form of delimiters. When we uncheck it, the generated regular expression will look like that in Figure D3, where each long expression, as shown in Figure 9, is replaced by \( \Sigma \) to simplify the representation. The "Display Code" check box controls whether to display the code of each character in the regular expression, as shown in Figure D4. "Range of delimiter number" controls the number of immediate delimiters between each pair of characters; e.g., when we set it to 2, then an IRI like "www.c--i--t--i-b-a.-n--k" cannot. "UC-SimList" controls the character–character similarity that you want to use.

Step 3. After generating the regular expression, we can use REGAP to detect phishing IRI/IDNs. We can simply copy and paste the IRI/IDNs for detection into the "IRI/IDN List" textbox in "Phishing IRI/IDN Detection" tab pad and click the "Detect>>" button. The detected potential phishing IRI/IDNs will be displayed in the "Detected List" textbox. Figure D5 shows that we have detected 10 phishing IRI/IDNs from 1656 IRI/IDNs.
In addition, we can edit the delimiters that we want to detect in the “Delimiter List” tabpad, one line for each delimiter. The default delimiters are the 22 legal symbols in IRI/IDN standard. The REGAP also uses the similar characters from UC-SimList for the regular expression generation.
FIGURE D3  Represent delimiter list with Σ.

FIGURE D4  Display the code of each character.
FIGURE D5  Phishing IRI/IDN detection.

FIGURE D6  Delimiter list.