Honey Encryption for Language
Robbing Shannon to Pay Turing?

Marc Beunardeau, Houda Ferradi, Rémi Géraud, and David Naccache

École normale supérieure, Information Security Group,
{first_name.given_name}@ens.fr

Abstract. Honey Encryption (HE), introduced by Juels and Ristenpart (Eurocrypt 2014, [13]), is an encryption paradigm designed to produce ciphertexts yielding plausible-looking but bogus plaintexts upon decryption with wrong keys. Thus brute-force attackers need to use additional information to determine whether they indeed found the correct key.
At the end of their paper, Juels and Ristenpart leave as an open question the adaptation of honey encryption to natural language messages. A recent paper by Chatterjee *et al.* [5] takes a mild attempt at the challenge and constructs a natural language honey encryption scheme relying on simple models for passwords.
In this position paper we explain why this approach cannot be extended to reasonable-size human-written documents *e.g.* e-mails. We propose an alternative solution and evaluate its security.

1 Introduction

Cryptography assumes that keys and passwords can be kept private. Should such secrets be revealed, any guarantee of confidentiality or authenticity would be lost. To that end, the set of possible secrets – the keyspace $\mathcal{K}$ – is designed to be very large, so that an adversary cannot possibly exhaust it during the system's lifetime.

In some applications however, the keyspace is purposely limited – for instance, passwords. In addition to the limited keyspace size, secret selection has a fundamental limitation: keys should be chosen uniformly at random – yet users routinely pick (the same) poor passwords. Consequently, key guessing is a guided process in which the adversary does not need to exhaust all possibilities. The deadly combination of low-entropy key generation and small keyspace make password-based encryption (PBE) particularly vulnerable [17].

The best security measure of a PBE is the min-entropy of the key distribution over $\mathcal{K}$:

$$\mu = -\log_2 \max_{k\in\mathcal{K}} p_k(k).$$

where p_k is the probability distribution of keys. The min-entropy captures how probable is the most probable guess. Conventional PBE schemes such as [**?**] can be broken with constant effort with probability $O(2^{-\mu})$, but μ is in practice very small: [2] reports $\mu < 7$ for passwords observed in a population of about

69 million users. If a message m were to be protected by such passwords, an adversary could easily recover m by trying the most probable passwords[1].

But how would the adversary *know* that the key she is trying is the correct one? A message has often some structure — documents, images, audio files for instance — and an attempt at decrypting with an incorrect key would produce something that, with high probability, does *not* feature or comply with this structure. The adversary can therefore tell apart a correct key from the incorrect ones, judging by how appropriate the decryption's output is. Mathematically, the adversary uses her ability to distinguish between the distribution of outputs for her candidate key k' and the distribution p_m of inputs she is expecting to recover.

Using such a distinguisher enables the attacker to try many keys, then select only the best key candidates. If there are not many possible candidates, the adversary can recover the plaintext (and possibly the key as well). In the typical case of password vaults, when one "master password" is used to encrypt a list of passwords, such an attack leads to a complete security collapse.

Example 1. Assume that we wish to AES-decrypt what we know is an English word protected with a small 4 digits key: $c \leftarrow \mathsf{Enc}_k(m)$. An efficient distinguisher is whether $m_{k'} \leftarrow \mathsf{Dec}_{k'}(c)$ is made of letters belonging to the English alphabet. For instance, if

$$c = \texttt{0f 89 7d 66 8b 4c 27 d7 50 fa 99 0c 5a d6 11 eb}$$

Then the adversary can distinguish between two candidate keys `5171` and `1431`:

$$m_{5171} = \texttt{48 6f 6e 65 79 00 00 00 00 00 00 00 00 00 00 00}$$
$$m_{1431} = \texttt{bd 94 11 05 a2 e5 a7 c8 48 57 87 2a 88 52 bc 7e}$$

Indeed, m_{5171} spells out '`Honey`' in ASCII while m_{1431} has many characters that do not correspond to any letters. Exhausting all 4 digit keys yields only one message completely made of letters, hence $k = \texttt{5171}$ and the adversary succeeded in recovering the plaintext m_{5171}.

To thwart such attacks, Juels and Ristenpart introduced Honey Encryption (HE) [13]. HE is an encryption paradigm designed to produce ciphertexts which, upon decryption with wrong keys, yields plausible-looking plaintexts. Thus brute-force attackers need to use additional information to decide whether they indeed found the correct key.

Mathematically, the decoding procedure in HE outputs candidate plaintexts distributed according to a distribution p_d close to the distribution p_m of real messages. This renders distinguishing attacks inoperant. The advantages of HE are discussed at length in [13] where the concept is applied to password-based encryption of RSA secret keys, credit card PINs and CVVs. In particular, HE does not reduce the security level of the underlying encryption scheme, but may act as an additional protection layer.

[1] Such passwords may be learnt from password leaks [12, 22, 23].

However, the applications of HE highlighted in [13] are very specific: Passwords protecting passwords (or passwords protecting keys). More precisely, low min-entropy keys protecting high min-entropy keys. The authors are wary not to extend HE to other settings and note that designing HE

"*...for human-generated messages (password vaults, e-mail, etc.) (...) is interesting as a natural language processing problem.*" [13]

To give a taste of the challenge, realizing HE as Juels and Ristenpart defined it is equivalent to modelling the probability distribution of human language *itself*. A more modest goal is to restrict to subsets of human activity where choices are more limited, such as passwords — this is indeed the target of a recent paper by Chatterjee, Bonneau, Juels and Ristenpart [5], which introduces encoders for human-generated passwords they call "natural language encoders" (NLE). Chatterjee *et al.*'s approach to language is to model the distribution of messages using either a 4-gram generative Markov model or a custom-trained probabilistic grammar model. This works reasonably well for passwords.

A natural question is therefore: Could the same techniques be extended or generalized to human-generated documents *in general*? Chatterjee *et al.* hint at it several times, but never actually take a leap: The core reason is that these approaches do not scale well and fail to model even simple sentences – let alone entire documents.

In this paper we give arguments why the approach of Chatterjee *et al.* does not extends, and give an alternative approach based on a corpus quotation distribution transforming encoding.

2 Preliminaries

Notations. We write $x \xleftarrow{D} X$ to denote the sampling of x from X according to a distribution D, and $x \xleftarrow{\$} X$ when D is the uniform distribution.

Message recovery attacks. Let $\mathcal{M}$ be a message space and let $\mathcal{K}$ be a key space. We denote by p_m the message distribution over $\mathcal{M}$, and by p_k the key distribution over $\mathcal{K}$. Let Enc be any encryption scheme. The *message-recovery advantage* of an adversary $\mathcal{A}$ against Enc is defined as

$$\mathbf{Adv}^{\mathrm{MR}}_{\mathsf{Enc},p_m,p_k}(\mathcal{A}) = \Pr\left[\mathrm{MR}^{\mathcal{A}}_{\mathsf{Enc},p_m,p_k} = \mathsf{True}\right]$$

where the MR security game is described in Game 1. $\mathcal{A}$ may run for an unbounded amout of time, and make an unbounded number of queries to a random oracle.

This advantage captures the ability of an adversary knowing the distributions p_m, p_k to recover a message encrypted with Enc.

When key and message entropy are low, this advantage might not be negligible. However, using Honey Encryption, Juels and Ristempart show that $\mathcal{A}$'s advantage is bounded by $2^{-\mu}$, where $\mu = -\log \max_{k \in \mathcal{K}} p_k(k)$ is the min-entropy of the key distribution.

Game 1 Message recovery (MR) security game $\mathrm{MR}^{\mathcal{A}}_{\mathsf{Enc},p_m,p_k}$.

```
$K' \xleftarrow{p_k} \mathcal{K}$
$M' \xleftarrow{p_m} \mathcal{M}$
$C' \xleftarrow{\$} \mathsf{Enc}(K', M')$
$M \leftarrow \mathcal{A}(C')$
return $M == M'$
```

Distribution Transforming Encoding. HE relies on a primitive called the *distribution transforming encoding* (DTE). The DTE is really the central object of HE, which is then used to encrypt or decrypt messages. A DTE is composed of two algorithms, $\mathsf{DTEncode}$ and $\mathsf{DTDecode}$ which map messages into numbers in the interval $[0, 1]$ and back, *i.e.* such that

$$\forall M \in \mathcal{M}, \quad \mathsf{DTDecode}(\mathsf{DTEncode}(M)) = M.$$

More precisely, $\mathsf{DTEncode}\colon \mathcal{M} \to [0, 1]$ is designed such that the output distribution of $\mathsf{DTEncode}$ is uniform over $[0, 1]$ when the input distribution over $\mathcal{M}$ is specified and known — in other terms, $\mathsf{DTDecode}$ samples messages in $\mathcal{M}$ according to a distribution p_d close to p_m, with

$$p_d(M) = \Pr\left[M' = M \mid x \xleftarrow{\$} [0, 1] \text{ and } M' \leftarrow \mathsf{DTDecode}(S)\right]$$

As such, DTEs cannot be arbitrary: They need to mimic the behaviour of the cumulative distribution function and its inverse. More precisely, the closeness of p_d and p_m is determined by the advantage of an adversary $\mathcal{A}$ in distinguishing the games of Figures 1 and 2:

$$\mathsf{Adv}^{\mathcal{A}}_{\mathsf{DTE},p_m} = \left|\Pr\left[\mathsf{SAMP1}^{\mathcal{A}}_{\mathsf{DTE},p_m} = 1\right] - \Pr\left[\mathsf{SAMP0}^{\mathcal{A}}_{\mathsf{DTE}} = 0\right]\right|$$

$\mathcal{A}$ is provided with either a real message and its encoding, or a fake encoding and its decoding. $\mathcal{A}$ outputs 1 or 0 depending on whether it bets on the former or the latter. A perfectly secure DTE is a scheme for which the indistinguishability advantage is zero even for unbounded adversaries (this is equivalent to $p_d = p_m$).

Fig. 1. $\mathsf{SAMP0}^{\mathcal{B}}_{\mathsf{DTE}}$

```
$x' \xleftarrow{\$} [0, 1]$
$M' \leftarrow \mathsf{DTDecode}(x')$
$b \xleftarrow{\$} \mathcal{B}(M', x')$
return $b$
```

Fig. 2. $\mathsf{SAMP1}^{\mathcal{B}}_{\mathsf{DTE},p_m}$

```
$M' \xleftarrow{p_m} \mathcal{M}$
$x' \xleftarrow{\$} \mathsf{DTEncode}(M')$
$b \xleftarrow{\$} \mathcal{B}(M', x')$
return $b$
```

Having good DTEs is the central aspect of building a Honey Encryption scheme as well as the main technical challenge. Given a good DTE, the honey encryption and decryption of messages is provided by a variation of the "DTE-then-encrypt" construction described in Figures 3 and 4 where some symmetric encryption scheme ($\mathsf{ESEncode}, \mathsf{ESDecode}$) is used. In the "DTE-then-encrypt" paradigm, a message is first transformed by the DTE into an integer x in some range, and x (or rather, some binary representation of x) is then encrypted with the key. Decryption proceeds by decrypting with the key, then reversing the DTE.

Fig. 3. Algorithm HEnc^{ES}

```
$\mathsf{HEnc}^{ES}(K, M)$
$x \leftarrow \mathsf{DTEncode}(M)$
$C \leftarrow \mathsf{ESEncode}(x, K)$
**return** $C$
```

Fig. 4. Algorithm HDec^{ES}

```
$\mathsf{HDec}^{H}(K, C)$
$x \leftarrow \mathsf{ESDecode}(K, C)$
$M \leftarrow \mathsf{DTDecode}(x)$
**return** $M$
```

2.1 Natural Language Encoding

Chaterjee *et al.* [5] developed an approach to generating DTEs based on two natural language models: an n-gram Markov model, and a custom probabilistic grammar tree.

Markov Model. The n-gram model is a local description of letters whereby the probability of the next letter is determined by the $n-1$ last letters:

$$\Pr[w_1 \cdots w_k] = \prod_{i=1}^{k} \Pr\left[w_i \mid w_{i-(n-1)} \cdots w_{i-1}\right]$$

It is assumed that these probabilities have been learnt from a large, consistent corpus.

Such models are language-independent, yet produce strings that mimic the local correlations of a training corpus — but, as Chomsky pointed out [6–8], the output of such models lacks the long-range correlations typical of natural language. The latter is not an issue though, as Chatterjee *et al.* train this model on passwords.

The model can be understood as a directed graph where vertices are labelled with n-grams, and edges are labelled with the cumulative probability from some distinguished root node. To encode a string it suffices to encode the corresponding path through this graph from the root — and decoding uses the input as random choices in the walk. Encoding and decoding can be achieved in time linear in message size.

Grammar Model. Probabilistic context-free grammars (PCFG) are language-dependent models that learn from a tagged corpus a set of grammatical rules, and then use these rules to generate syntactically possible sentences. PCFGs are a compact way of representing a distribution of strings in a language.

Although it is known that context-free grammars do not capture the whole breadth of natural language, PCFGs are a good starting point, for such grammars are easy to understand, and from a given probabilistic context-free grammar, one can construct compact and efficient parsers [16]. The Stanford Statistical Parser, for instance, has been used by the authors to generate parse trees in this paper.

Mathematically, a probabilistic context-free grammar G is a tuple of the form $(N, T, R, P, \textsc{Root})$ where N are non-terminal symbols, T are terminal symbols (disjoint from N), R are production rules, P is the set of probabilities on production rules and Root is the start symbol. Every production rule is of the form $A \to b$, where $A \in N$ and $b \in (T \cup N)^*$.

Figure 5 shows a parse tree aligned with a sentence. Some grammatical rules can be read at every branching: S $\to$ NP VP, NP $\to$ DT VBN NN, NP $\to$ DT NN, *etc.*

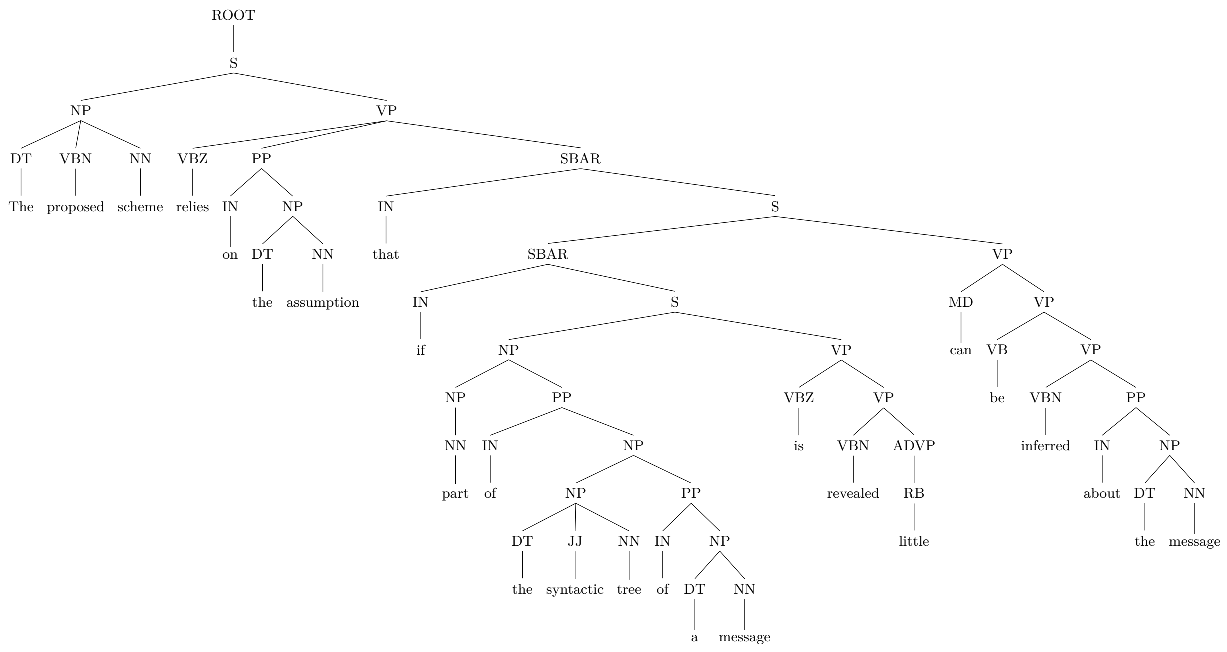


Fig. 5. Syntactic tree of an example sentence.

Chaterjee *et al.* [5] rely on a password-specific PCFGs [12,15,18,22,23] where grammatical roles are replaced by *ad hoc* roles.

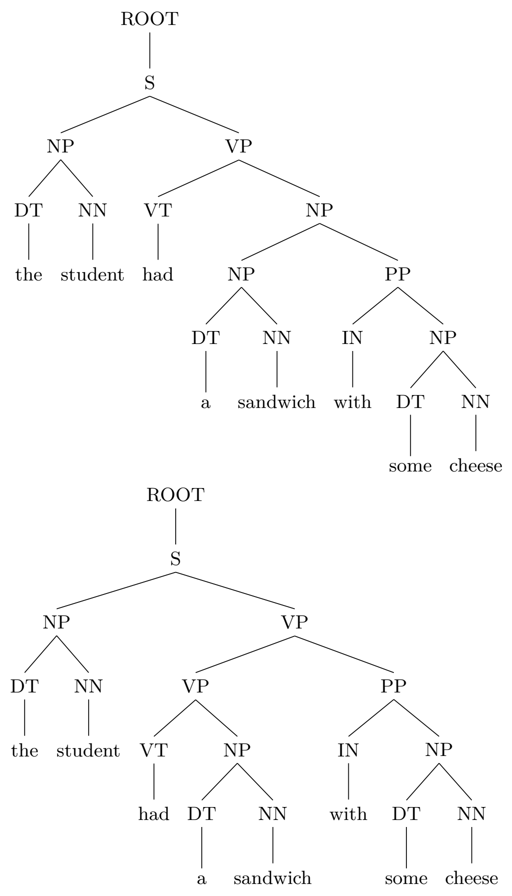


Fig. 6. Two possible derivations of the same sentence. Note that these derivations correspond to two possible meanings which are not identical.

The DTE encoding of a string is the sequence of probabilities defining a parse tree that is uniformly selected from all parse trees generating the same string (see *e.g.* Figure 6, which provides an example of two parse trees for a same sentence, amongst more than 10 other possibilities). Decoding just emits the string indicated by the encoded parse tree.

In the probabilistic context, the probability of each parse tree can be estimated. A standard algorithm for doing so is due to Cocke, Younger, and Kasami (CYK) [9,14,24].

Generalized Grammar model. The generalized idea relies on the assumption that if part of the syntactic tree of a message is revealed, little can be inferred about the message. To understand the intuition, consider the syntactic tree of the previous sentence (clause) shown in Figure 7.

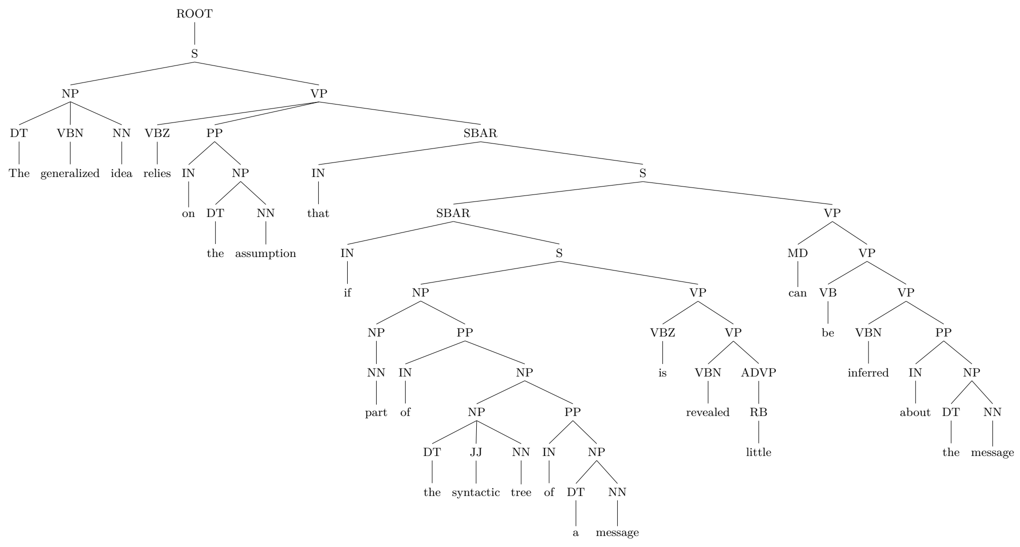


Fig. 7. Syntactic tree of an example sentence.

As we can see, words are tagged using the *clause level*, *phrase level* and *word level* labels listed in Appendix A.

The idea underlying syntactic honey encryption consists in revealing a rewritten syntactic tree's word layer while encrypting words[2]. The process starts by a syntactic analysis of the message allowing to extract the plaintext's syntactic tree. This is followed by a projection at the word level. When applied to the previous example, we get the projection denoted by S (hereafter called *skeleton*):

$$S = \text{DT VBN NN VBZ IN DT NN IN IN NN IN DT JJ}$$
$$\text{NN IN DT NN VBZ VBN RB MD VB VBN IN DT NN}$$

Given a clause, we can automatically associate each word s_i to a label L_i[3]. For instance, if the third word of the clause is "relies", then $L_3 \leftarrow \text{VBZ}$. We denote by R_i the rank of the skeleton's i-th word in the dictionary of the category L_i. Finally we denote by $|X|$ the cardinality of the set X.

To map our ordered wordlist into a single integer, we note that because in the above example there are 5 DTs, 3 VBNs, 6 NNs, 2 VBZs, 6 INs, and 1 JJ, RB, MD and 1 VB, our specific clause is one amongst exactly B syntactically correct messages where:

$$B = |\text{DT}|^5 |\text{VBN}|^3 |\text{NN}|^6 |\text{VBZ}|^2 |\text{IN}|^6 |\text{JJ}||\text{RB}||\text{MD}||\text{VB}|$$

[2] We stress that unlike e.g. Kamouflage [1] which deals with passwords, syntactic honey encyrption applies to natural language.

[3] Note that such a skeleton might be ambiguous in certain constructions, for instance in sentences such as "*Time flies like an arrow; fruit flies like a banana*".

```
Decoding
$\ell \leftarrow |L_0|$
for $i \leftarrow 0$ to $k-1$
    $R_i \leftarrow e \bmod \ell$
    $e \leftarrow (e - R_i)/\ell$
    $\ell \leftarrow \ell \times |L_{i+1}|$
    $\text{word}_i \leftarrow \text{Dictionary}_{L_i}(R_i)$
```

Fig. 8. Decoding algorithm.

We can thus map a clause skeleton into $\mathbb{N}$ by writing:

$$e \leftarrow \sum_{i=0}^{k-1} R_i \prod_{j=0}^{i-1} |L_j|$$

where, by typographic convention, $L_{-1} = 1$.

To get back the original clause, given e and the skeleton, we use the algorithm of Figure 8.

The skeleton is transferred in clear:

$$s = \text{DT VBN NN VBZ IN DT NN IN IN NN IN DT JJ NN}$$
$$\text{IN DT NN VBZ VBN RB MD VB VBN IN DT NN}$$

Note that there is no need to tune precisely the plaintext size of the underlying block cipher because the decoding process for e stops automatically when i reaches $k-1$. In other words, we can randomize encryption at little cost by replacing e by $e + \mu B$ for some random integer μ.

The number e is then honey encrypted, thus attempting to protect the actual content of the plaintext sentence.

3 Limitations of Honey Encryption

As observed by [13], HE security is threatened when $\mathcal{A}$ has some side information about the target message. This puts strong constraints on HE's applicability to situations such as protecting RSA or HTTPS private keys. A second limitation is that the HE construction assumes that the key and message distributions are independent. When these distributions are correlated, $\mathcal{A}$ can identify a correct message by comparing that message with the decryption key that produced it. Similarly, encrypting two correlated messages under the same key enables $\mathcal{A}$ to identify correct messages.

Finally, constructing a DTE requires knowing the distribution p_m of messages in $\mathcal{M}$. As we will argue, this turns out to be extremely difficult to evaluate when $\mathcal{M}$ becomes a large enough space, such as human-generated messages (emails, *etc.*). In those cases, it might even turn out that adversaries know p_m *better than users*.

The methods described in Section 2.1 apply reasonably well to short passwords, but as we will now argue they cannot scale to deal with natural language as used in real-world scenarios such as: e-mails and written documents. The reason is threefold: First the methods require a huge amount of context-relevant information; Second, even when this information is available, the methods of [5] fail to produce convincing honey messages, *i.e.* messages that fool *automated tools* in telling them apart from real messages with high probability; Third, natural language HE may actually leak information about the underlying message.

Scaling NLE. The models developed for passwords in [5] can be extended: Markov models for instance can be configured to generate arbitrary-length messages. Instead of letters, such models can be trained to produce words, in accordance with some known distribution of n-grams. But while there are only a few English letters, a recent study of the English language [20] counts more than a million individual words in usage.

As a result assuming we use one hundredth of the English language, the memory required to store an n-gram database is of the order of $10^{4n} \approx 2^{13n}$. That becomes a problem not only in terms of storage, but also when access latency is taken into account. Applying directly the method of [5] to words (using $n = 5$) would require knowing, storing, and sharing 2^{65} bytes of data[4]. The real issue however is that measuring accurately 5-grams usage is extremely difficult in practice, so that most of this impossibly large database is essentially unknown[5].

Using grammars is one way to avoid this combinatorial explosion by keeping a simple and compact model of language. To that end, a sentence is parsed to reveal its grammatical structure as in Figures 5 and 6. Each word is labelled with an indication of its grammatical role (see Appendix A).

A sentence is therefore uniquely represented by a list of grammatical tags, and a list of integers denoting which word is used. The idea behind syntactic honey encryption consists in revealing the tags but honey encrypting the words. By construction, generated honey messages have the same syntax as the original message, which makes decryption with a wrong key yield an often *plausible* plaintext. For instance, a sentence such as s_1 = "Secure honey encryption is hard" could be honey decrypted as Chomsky's famous sentence s_2 ="Colorless green ideas sleep furiously" [6], illustrating a sentence that is grammatically correct while being semantically void. Here s_1 and s_2 share the same syntax. To use this algorithm the communicating parties must agree on a dictionary that includes a set of labels and a parsing algorithm.

There are however two structural limitations to this grammatical approach. First, revealing the syntactic structure of a message leaks information. This is a very big deviation from classical cryptography, since it has always been taken for granted -for obvious reasons- that a ciphertext should not leak anything

[4] This is conceptually similar to Borges' famous library [3, 4].
[5] See for instance http://www.ngrams.info/.

but the length of the underlying plaintext. On a more practical note unless the message is long enough, there might be only very few possible sentences with that given syntax. Second, a grammar is language-dependent — and furthermore, to some extent, there is variability within a given language[6]. The consequence of an inaccurate or incorrect tagging is that upon honey decoding, the sentence might be noticeably incorrect from the suitable linguistic standpoint.

This opens yet another research avenue. Automatically translate the sentence into an artificially created language where syntactic honey encryption would be very efficient. For instance translate French to Hindi, then perform honey encryption on the Hindi sentence.

Quality of NLE. The question of whether a honey message is "correct" in a given linguistic context can be rephrased: Is it possible, to an adversary having access to a large corpus (written in the same language), to distinguish honey messages from the legitimate plaintext?

It turns out that the two approaches to modelling natural language provide two ways to construct a distinguisher: We can compare a candidate to a reference, either statistically or syntactically. But we can actually do both *simultaneously*: We can use Web search engines to assess how often a given sentence or word is used[7]. This empirical measure of probability is interesting in two respects: First, an adversary may query many candidates and prune those that score badly; Second, the sender cannot learn enough about the distribution of *all* messages using that "oracle" to perform honey encryption.

The situation is that there is a measurable distance between the model (used by the sender) of language, and language itself (as can be measured by *e.g.* a search engine). Mathematically, the sender assumes an approximate distribution p_m on messages which is different from the real-world distribution $\widehat{p}_m$. Because of that, a good DTE in the sense of Figures 1 and 2 would, in essence, yield honey messages that follow p_m and not $\widehat{p}_m$. An adversary capable of distinguishing between these distributions can effectively tell honey messages apart.

What is the discrepancy between p_m and $\widehat{p}_m$? Since $\widehat{p}_m$ measures real-world usage, we can make the hypothesis that such messages correspond to human concerns, *i.e.* that they carry some meaning — in one word, what distinguishes p_m from $\widehat{p}_m$ is *semantics*.

Leaking information. Another inherent limitation of HE is precisely that decryption of uniformly random ciphertexts produces in general the most probable messages. There are many situations in which linguistic constraints force a certain structure on messages, *e.g.* the position of a verb in a German sentence.

[6] An extreme example is William Shakespeare's use of inversion as a poetic device: "*If't be so, For Banquo's issue have I fil'd my mind,/ For them the gracious Duncan have I murther'd,/Put rancors in the vessel of my peace*" (*MacBeth*, III.1.8).

[7] We may assume that communication with such services is secure, *i.e.* confidential and non-malleable, for the sake of argument.

Consequently, there might be enough landmarks for a meaningful reconstruction (see also [21]).

To thwart such reconstruction attacks, it is possible to consider phrase-level defences. Such defences imply modifying the syntactic tree in a way which is both reversible and indistinguishable from other sentences of the language. Phrase-level defences heavily depend on the language used. For instance the grammar of Latin, like that of other ancient Indo-European languages, is highly inflected; consequently, it allows for a large degree of flexibility in choosing word order. For example, *femina togam texuit*, is strictly equivalent to *texuit togam femina* or *togam texuit femina*. In each word the desinence (also called ending or suffix): *-a*, *-am* and *-uit*, and not the position in the sentence, marks the word's grammatical function. This specific example shows that even if the target language allows flexibility in word order, this flexibility does not necessarily imply additional security. Semitic languages, such as Arabic or Hebrew, would on the contrary offer very interesting phrase-level defences. In semitic languages, words are usually formed by associating three, four or five-consonant verbs to structures. In Hebrew for example the structure *mi□□a□a* corresponds to the place where action takes place. Because the verb *drš* means *to teach* (or *preach*), and because the verb *zrk* means *to throw* (or *project*), the words *midraša*[8] and *mizraka* respectively mean "school" and "water fountain" (the place that projects (water)). This structure which allows, in theory, to build $O(ab)$ terms using $O(a)$ verbs and $O(b)$ and thus turns out to be HE-friendly.

4 Corpus Quotation DTE

We now describe an alternative approach which is interesting in its own right. Instead of targeting the whole breadth of human language, we restrict users to only quote from a known public document[9].

The underlying intuition is that, since models fail to capture with enough detail the empirical properties of language, we should think the other way around and start from an empirical source directly. As such, the corpus quotation DTE addresses the three main limitations of HE highlighted in Section 3: It scales, it produces realistic sentences (because they are actual sentences), and it does not leak structural information.

Consider a known public string $\mathfrak{M}$ (the "corpus"). We assume that $\mathcal{M}$ consists in contiguous sequence of words sampled from $\mathfrak{M}$, *i.e.* from the set of substrings of $\mathfrak{M}$. To build a DTE we consider the problem of mapping a substring $m \in \mathcal{M}$ to $[0, 1]$.

Interval encoding of substrings. Let M be the size of $\mathfrak{M}$, there are $|\mathcal{M}| = M(M-1)/2$ substrings denoted $m_{i,j}$, where i is the starting position and j is the ending position, with $i \leq j$. Substrings of the form $m_{i,i}$ are 1-letter long.

[8] The Arabic equivalent is *madrasa*.

[9] The way some characters do in Umberto Eco's novel, *Il pendolo di Foucault* [10].

The DTE encoding of $m \in \mathcal{M}$ is a point in a sub-interval of $[0, 1]$, whose length is proportional to the probability $p_m(m)$ of choosing m. If p_m is uniform over $\mathcal{M}$, then all intervals have the same length and are of the form

$$I_k = \left] \frac{2k}{M(M-1)}, \frac{2(k+1)}{M(M-1)} \right].$$

where k is the index of $m \in \mathcal{M}$ for some ordering on $\mathcal{M}$. Decoding determines which I_k contains the input and returns k, from which the original substring can be retrieved. For more general distributions p_m, each substring $m_{i,j}$ is mapped to an interval whose size depends on $p_m(m)$.

Length-dependent distributions. Let's consider the special case where $p_m(m)$ depends only on the length of m. We will therfore consider the function $p : [1, M] \longrightarrow [0, 1]$ giving the probability of a substring of a given length. This captures some properties of natural languages such as Zipf's law [19]: Short expressions and words are used much more often than longer ones. Note that part of this is captured by the fact that there are fewer long substrings than short ones.

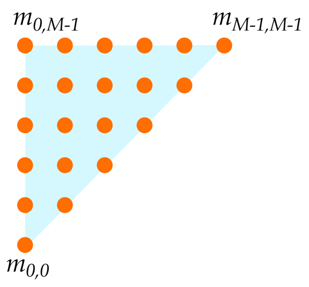


Fig. 9. Triangle representation T of the substrings $\mathcal{M} \subseteq \mathfrak{M}$. Substrings along right diagonals have equal length. The top-left point represents the entire corpus $\mathfrak{M}$.

Thus the encoding of a message $m_{i,j}$ is a random point in an interval of size $\ell(j - i)$ proportional to $p_m(m_{i,j}) = p(j - i)$:

$$\ell(k) = \frac{p(k)}{L}, \qquad L = \sum_{k=1}^{M} (M - k)p(k).$$

This ensures that

$$\sum_{k=1}^{M} (M - k)\ell(k) = 1.$$

The intervals associated to each substring are defined as follows. First, substrings $m_{i,j}$ are mapped via the map $\tau\colon m_{i,j} \mapsto (i,j)$ to a triangle (see Figure 9):

$$T = \{(i,j) \mid j \geq i \in [0, M-1]\} \subset \mathbb{N}^2.$$

Then points in T are mapped to $[0,1]$ using the function:

$$\Phi\colon (i,j) \mapsto (i-1)\ell(\operatorname{diag}(i,j)) + \sum_{k=1}^{\operatorname{diag}(i,j)-1} k\ell(k)$$

where $\operatorname{diag}(i,j) = M-1-(j-i)$ indicates on which upright diagonal (i,j) is. All in all, a substring $m_{i,j}$ is encoded using the following algorithm:

$$\mathsf{DTEncode} : m_{i,j} \mapsto (\Phi + \epsilon\ell \circ \operatorname{diag})\,(\tau(m_{i,j}))$$

where ϵ is sampled uniformly at random from $[0,1]$.

Encoding can be understood as follows: Substrings of equal length k are mapped by τ to points along a diagonal of constant $k = j-i$. The first diagonal is the whole corpus $\mathfrak{M}$ and the only substring of length M. The $(M-1-k)$-th diagonal is the set of substrings $\{m_{i,i+k} \mid i \in [0, M-1-k]\}$ of length k. Decoding is achieved by Algorithm 1, which takes a number $x \in [0,1]$ and returns the position $(i,j) = \Phi^{-1}(x)$ of the corresponding substring by determining the position in T. The idea is to count the segment length before x. At each iteration we update the segment length and the current position in the diagonal.

Algorithm 1 Position of $\Phi^{-1}(x)$

```
Input: $x \in [0,1]$
Output: $(a,b) \in [|0,M|]^2$ such that $\Phi(a,b) = x$
  $i \leftarrow 0$
  $j \leftarrow 0$
  $k \leftarrow M$
  while $i < x$ do
    $i \leftarrow i + \ell(k)$
    $j \leftarrow j + 1$
    if $j \geq M - k + 1$ then
      $j \leftarrow 0$
      $k \leftarrow k - 1$
    end if
    return $(j-1, M+j-k-1)$
  end while
```

This decryption algorithm is linear in the number of substrings, *i.e.* it runs in time $O(M^2)$. We can speed things up using pre-computations, Algorithms 2 and 3 run in $O(M)$ time and memory.

Algorithm 2 Pre-computation

```
Output: vector $V$ such that intervals in $[V[i], V[i+1]]$ are the intervals of length $\ell(i)$
  let $V[1..M]$ be a vector of length $M$
  for $i \leftarrow 1$ to $M$ do
    $V[i] \leftarrow V[i-1] + (M-i+1)\ell(i)$
  end for
  return $V$
```

Algorithm 3 Fast Decryption

```
Input: $x \in [0,1], V$ the result of Algorithm 2.
Output: $(a,b) \in [|0,M|]^2$ such that $\Phi(a,b) = x$
  $i \leftarrow 1$
  while $V[i] < x$ do
    $i++$
  end while
  $j \leftarrow (x - V[i])/\ell(i)$
  return $(j-1, M-i-1)$
```

5 Further Research

This work opens a number of interesting research directions:

Machine to Human HE: Search engines, and more generally computational knowledge engines and answer engines such as Wolfram Alpha[10] provide users with structured answers that mimic human language. These algorithms generate messages using well-defined algorithmic process having a precise probability distribution which DTEs can be better modelled. Such sentences are hence likely to be safer to honey encrypt.

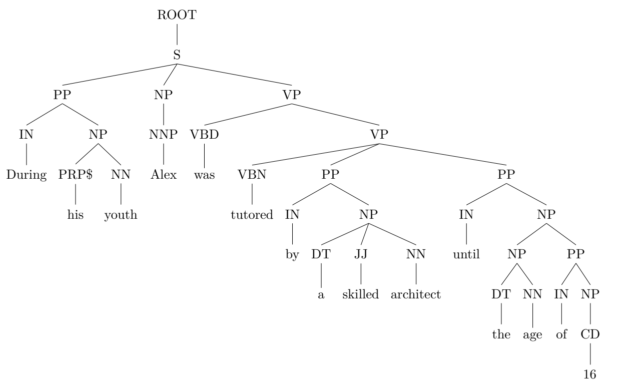
Automated Plaintext Pre-Processing: A more advanced, yet not that unrealistic option consists in having a machine understand a natural language sentence m and re-encode m as a humanly understandable yet grammatically and syntactically simplified sentence m' having the same meaning for a human. Such an ontology-preserving simplification process will not modify the message's meaning while allowing the construction better DTEs.

Adding Syntactic Defenses: This work was mostly concerned by protecting messages at the *word* level. It is however possible to imagine the adding of defenses at the clause and at the phrase levels. Two simple clause-level protections consist in adding decoy clauses to the message, and shuffling the order of clauses in the message. Both transforms can be easily encoded in the ciphertext by adding an integer field interpreted as the rank of a permutation and a binary strong

[10] www.wolframalpha.com.

whose bits indicate which clauses should be discarded. Decryption with a wrong key will yield a wrong permutation and will remove useful skeletons from the message. It should be noted that whilst the permutation has very little cost, the addition of decoy skeletons impacts message length. It is important to use decoy skeletons that are indistinguishable from plausible skeletons. To that end the program can either pick skeletons in a huge database (*e.g.* the web) or generate them artificially.

Adding Phrase-Level Defenses: Adding phrase-level defenses is also a very interesting research direction. A simple way to implement phrase-level defenses consists in adding outgrowths to the clause. An outgrowth is a collection of fake elements added using a specific rewriting rule. Note that information cannot be removed from the sentence. Here is an example of scrambling using outgrowths: the original clause m_0 is the sentence “During his youth Alex was tutored by a *skilled* architect until the age of 16”. The syntactic tree of m_0 is:



The skeleton of m_0 is IN PRP$ NN NNP VBD VBN IN DT JJ NN IN DT NN IN CD.

Now consider the following rewriting rules:

PRP$ NN $\longrightarrow$ PRP$ JJ NN

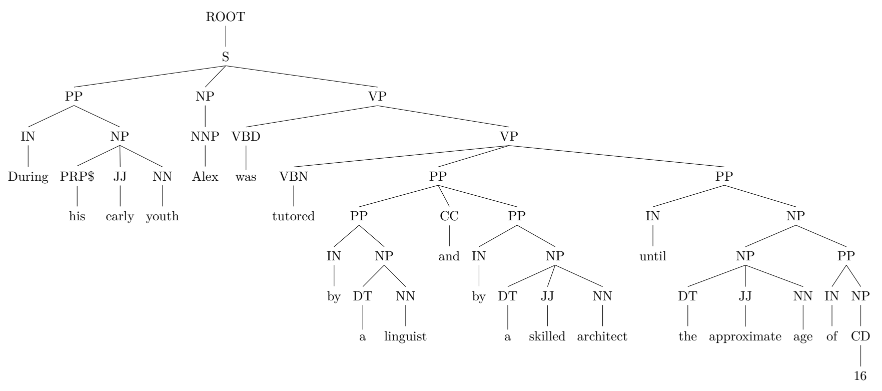
DT NN $\longrightarrow$ DT JJ NN

IN DT JJ NN $\longrightarrow$ IN DT NN CC IN DT JJ NN

We can apply these rules to m_0 to obtain:

m_0	IN PRP$ NN NNP VBD VBN IN DT JJ NN IN DT NN IN CD
$m_1 \leftarrow r_1(m_0)$	IN PRP$ JJ NN NNP VBD VBN IN DT JJ NN IN DT NN IN CD
$m_2 \leftarrow r_2(m_1)$	IN PRP$ JJ NN NNP VBD VBN IN DT JJ NN IN DT JJ NN IN CD
$m_3 \leftarrow r_3(m_2)$	IN PRP$ JJ NN NNP VBD VBN IN DT NN CC IN DT JJ NN IN DT JJ NN IN CD

m_3 is a plausible skeleton that could have corresponded to the clause: “During his early youth Alex was tutored by a linguist and by a skilled architect until the approximate age of 16”:



It remains to show how to reverse the process to recover the original skeleton m_0. To that end, we include in the ciphertext a binary string indicating which outgrowths should be removed. Removal consists in scanning m_0 and identifying what could have been the result of rewriting. Scanning reveals one potential application of rule 1 (namely "his early youth"), two potential applications of rule 2 ("a skilled architect" and "the approximate age") and one potential application of rule 2 ("by a linguist and by a skilled architect"). Hence 4 bits suffice to identify and remove the outgrowths.

References

1. Bojinov, H., Bursztein, E., Boyen, X., Boneh, D.: Kamouflage: Loss-resistant password management. In: Computer Security–ESORICS 2010, pp. 286–302. Springer (2010)
2. Bonneau, J.: The science of guessing: Analyzing an anonymized corpus of 70 million passwords. In: IEEE S&P 2012 [11], pp. 538–552
3. Borges, J.L.: El Jardín de senderos que se bifurcan. Editorial Sur (1941)
4. Borges, J.L.: Ficcione. Editorial Sur (1944)
5. Chatterjee, R., Bonneau, J., Juels, A., Ristenpart, T.: Cracking-resistant password vaults using natural language encoders. In: 2015 IEEE Symposium on Security and Privacy. pp. 481–498. IEEE Computer Society Press, San Jose, California, USA (May 17–21, 2015)
6. Chomsky, N.: Three models for the description of language. Information Theory, IRE Transactions on 2(3), 113–124 (1956)
7. Chomsky, N.: On certain formal properties of grammars. Information and control 2(2), 137–167 (1959)
8. Chomsky, N.: Syntactic structures. Walter de Gruyter (2002)
9. Cocke, J.: Programming languages and their compilers: Preliminary notes (1969)
10. Eco, U.: Il pendolo di Foucault. Bompiani (2011)
11. 2012 IEEE Symposium on Security and Privacy. IEEE Computer Society Press, San Francisco, California, USA (May 21–23, 2012)
12. Jakobsson, M., Dhiman, M.: The benefits of understanding passwords. In: Traynor, P. (ed.) 7th USENIX Workshop on Hot Topics in Security, HotSec'12, Bellevue, WA, USA, August 7, 2012. USENIX Association (2012)

13. Juels, A., Ristenpart, T.: Honey encryption: Security beyond the brute-force bound. In: Nguyen, P.Q., Oswald, E. (eds.) Advances in Cryptology – EUROCRYPT 2014. Lecture Notes in Computer Science, vol. 8441, pp. 293–310. Springer, Heidelberg, Germany, Copenhagen, Denmark (May 11–15, 2014)
14. Kasami, T.: An efficient recognition and syntax analysis algorithm for context-free languages. Tech. rep., DTIC Document (1965)
15. Kelley, P.G., Komanduri, S., Mazurek, M.L., Shay, R., Vidas, T., Bauer, L., Christin, N., Cranor, L.F., Lopez, J.: Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms. In: IEEE S&P 2012 [11], pp. 523–537
16. Klein, D., Manning, C.D.: Accurate unlexicalized parsing. In: Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1. pp. 423–430. Association for Computational Linguistics (2003)
17. Li, Z., He, W., Akhawe, D., Song, D.: The emperor's new password manager: Security analysis of web-based password managers. In: Fu, K., Jung, J. (eds.) Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014. pp. 465–479. USENIX Association (2014)
18. Ma, J., Yang, W., Luo, M., Li, N.: A study of probabilistic password models. In: 2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014. pp. 689–704. IEEE Computer Society (2014)
19. Manning, C.D., Schütze, H.: Foundations of statistical natural language processing. MIT Press (2001)
20. Michel, J.B., Shen, Y.K., Aiden, A.P., Veres, A., Gray, M.K., Pickett, J.P., Hoiberg, D., Clancy, D., Norvig, P., Orwant, J.: Quantitative analysis of culture using millions of digitized books. Science 331(6014), 176–182 (2011)
21. Rayner, K., White, S.J., Johnson, R.L., Liversedge, S.P.: Raeding wrods with jubmled lettres there is a cost. Psychological science 17(3), 192–193 (2006)
22. Veras, R., Collins, C., Thorpe, J.: On semantic patterns of passwords and their security impact. In: ISOC Network and Distributed System Security Symposium – NDSS 2014. The Internet Society, San Diego, California, USA (Feb 23–26, 2014)
23. Weir, M., Aggarwal, S., de Medeiros, B., Glodek, B.: Password cracking using probabilistic context-free grammars. In: 2009 IEEE Symposium on Security and Privacy. pp. 391–405. IEEE Computer Society Press, Oakland, California, USA (May 17–20, 2009)
24. Younger, D.H.: Recognition and parsing of context-free languages in time n^3. Information and Control 10(2), 189–208 (1967)

A Grammatical tags for English

Table 1. Partial list of grammatical roles.

	Clause Level
S	Simple declarative clause
SBAR	Clause introduced by a (possibly empty) subordinating conjunction.
	Phrase Level
ADVP	Adverb phrase
NP	Noun phrase
PP	Prepositional phrase
VP	Verb phrase
	Word Level
CC	Conjunction, coordinating
DT	Determiner
IN	Preposition or subordinating conjunction
JJ	Adjective
MD	Modal
NN	Noun, singular or mass
PRP	Pronoun, personal
PRP$	Pronoun, possessive
RB	Adverb
VB	Verb, base form
VBN	Verb, past participle
VBZ	Verb, third person singular present