Alice and Bob's life stories

Cryptographic communication using shared experiences

Joseph Bonneau

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

Abstract. We propose a protocol for confidential one-way communication between two parties who know each other well using only pre-existing knowledge from their shared life experience. This could enable, for example, lovers or close friends to communicate without prior key exchange. Our system uses a flexible secret-sharing mechanism to accommodate personal knowledge of variable guessing resistance and memorability with reasonable overhead in terms of computation and storage.

1 Introduction

Traditional cryptographic communication relies on artificially-created high entropy random strings as secret keys. In some cases, two parties who know each other well may already possess enough secret knowledge in the form of shared experiences to enable secure communication. For example, a husband and wife will both remember many details of their courtship that no other parties know, while a group of siblings can recall minutia of their childhood which are unknown outside of the family.

We explore cryptographic communication using this naturally-shared secret knowledge in situations where it is otherwise not possible to share a cryptographic key. Our goal is to design a secure protocol for encryption over a one-way communication channel. Abstractly, Alice wishes to send Bob a message, and she can only use her shared experiences with Bob to ensure confidentiality and integrity of the message. This will be achieved with a secure protocol for converting answers to personal questions into a high-entropy key suitable for conventional encryption.

We assume that Alice must send her message urgently without actually meeting Bob to exchange key material. This may be because the message is being sent for future decryption by Bob. For example, Alice may be writing a will that she wants to keep hidden in her lifetime. By encrypting it using personal secrets shared with Bob, only he can read it after her death but he does not need to be made aware of it by an explicit key transfer. Another example is encryption of backup key material intended for oneself in the future, in the case of accidentally losing a stored key or forgetting a password. This is the example which has motivated previous research [1,2], though we consider this example as a special case of a more general concept.

2 Related work

2.1 Personal knowledge questions

Most experience in designing personal-knowledge questions (PKQs) has been in the context of backup authentication in the case of forgotten passwords. The classic example is a bank which requires a user to enter his or her mother's maiden name to reset a password or PIN. Just wrote a survey of PKQs in the context of web re-authentication which enumerated the many different design parameters from a usability standpoint [3].

Our application is much more demanding than online authentication, where it is possible to limit the number of guess an adversary can make. Still, it will be critical to have some understanding of the security and memorability of PKQs, since they are our primary means of turning shared knowledge into cryptographic keys. The few formal studies conducted have found that questions typically used in practice have very poor security characteristics. Rabkin surveyed PKQ systems used in real banking websites, and reported that a large number of questions suffered from either poor usability or vulnerability to mining data available on the web [4]. An earlier study by Griffith and Jakobsson discovered exactly the mother's maiden name of over 15% of Texas residents using publicly available data [5]. Concurrent research in the field of social network security has found that simply knowing a victim's list of friends can enable inference of private information such as gender or political beliefs [6,7], making the secure choice of PKQs difficult.

Jakobsson et al proposed "preference-based authentication" to decrease the reliance on publicly available information by using questions such as "do you like country music?" [8]. They later refined their system and conducted extensive user studies [9] and found that preferences are much more difficult to mine from the Internet, while still being highly memorable.

2.2 Password backup systems

We are inspired by two previously-proposed schemes for cryptographically secure password backup. Ellison et al. proposed a scheme in 1999 for encryption using "Personal Entropy" [1]. This system involved hashing a set of answers to stored PKQs into a set of shares of a secret, and using an (n, t)-threshold secret-sharing scheme to recover a master key.

Frykholm and Juels proposed another scheme in 2001 [2], based on fuzzy commitment, which relied on mapping a user's set of answer to PKQs to the message space of an error-correcting code, then decoding to recover the master key, which was chosen at random from the space of valid codewords. This scheme is provably secure (under certain assumptions about the coding system) in that an attacker's probability of successfully guessing the correct master key is bounded by the min-entropy of the set of possible answers.

3 Desired properties

Either of the password backup schemes describe in Section 2.2 could be adapted for our desired scenario of communication between two different parties. However, we would like to add several new features.

3.1 Variable security of questions

Previous schemes haven't been designed to combine PKQs of different strength levels. We may wish to encrypt based on the answers to several questions whose answer is a name (*e.g.* "What was our neighbour's oldest son's name growing up?") in addition to many true-false or multiple-choice questions, which are easier to create but also easier to guess. Previous schemes map all questions into equally sized blocks to make design simpler. We will design for questions with answer-spaces of different sizes, considering this a desired property for a secure and usable system.

3.2 Variable memorability of questions

Another inflexibility in the previous schemes is the assumption that all answers have an equally likely chance of being forgotten or not known at the time of decryption. This is reflected in that errors in the response to any question are treated equally by the error correction scheme. We wish to efficiently accommodate questions whose recall probabilities vary. For example, a sender may be close to 100% certain that the recipient can answer a certain question, and thus not provide any error tolerance for this question and force an adversary to guess it correctly. Similarly, some questions may be known to be likely to be forgotten, and will require extra error tolerance to ensure the system is usable.

3.3 Key strengthening

Since we are limited by the ability of our user interface to extract memorable and unpredictable answers, we can use key strengthening [10] to increase the difficulty of brute-force guessing of possible answers. This is simple in our application since our system already consists of converting answers to PKQs into a master key. We will add the extra step of hashing our pre-master key 2^s times before it is used to decrypt message content. This effectively adds s bits of work for an attacker attempting to guess the correct answers. For $s \sim 10$–20, it will add only a small amount of slowdown to decryption using a desktop computer, which is acceptable since our system is expected to be used only in rare circumstances.

3.4 Sacrificed properties

In order to achieve the above properties, we are willing to sacrifice several performance properties, considering our system to be a rarely-used emergency protocol. In particular, we are willing to add considerable storage overhead to an encrypted message, and slowdown in encrypting and decrypting. We will examine a sample requirement of storage and slowdown in Section 4.5.

4 Proposed system

4.1 Protocol description

Encryption under our system will consist of choosing a set Q of questions $\{q_0, \ldots, q_m\}$, and providing a corresponding set of answers $A = \{a_0, \ldots, a_m\}$. We then randomly pick a pre-master key K_{P}, and compute the master encryption key according to our desired level of key-strengthening:

$$K_{\mathrm{M}} = \mathbf{H}^{2^s}(K_{\mathrm{P}}) \tag{1}$$

The critical step in our protocol which is different from previous systems is that encryption will require enumerating subsets of A which allow for recovery of K_{M}. We will not limit the system to subsets of a specific size, but allow for total flexibility. To achieve this, encryption will require explicitly listing subsets $A_i \in A$ along with decryption information. We denote the set of sets of acceptable answers as

$$A^* = \{A_i \in A : \text{knowledge of } A_i \text{ shall enable decryption}\} \tag{2}$$

We will use a trivial secret-sharing scheme to accomplish this, namely, using exclusive-or to combine hashes of answers to derive a subset key for each $A_i \in A^*$:

$$K_i = \bigoplus_{a_j \in A_i} \mathbf{H}(a_j || j) \tag{3}$$

We include the question number j in each hash to force the adversary to search over all answers to all questions in case some answers may possible for multiple questions. For each subset key K_i we then compute the offset O_i which allows us to recompute the pre-master key:

$$O_i = K_{\mathrm{P}} \oplus K_i \tag{4}$$

To send a message M, Alice transmits:

$$\mathbf{AE}_{K_{\mathrm{M}}}(M||A),\ Q,\ O \tag{5}$$

We use a standard symmetric-key authenticated encryption function **AE**, such as AES-GCM, to ensure confidentiality and integrity. We note that Alice must transmit the offset O_i for every answer set $A_i \in A^*$. We will examine the storage requirements in a sample deployment in Section 4.5. We also note that Alice appends the correct answers to her questions to the plaintext before encrypting, this will be discussed in Section 6.1.

Given correct receipt of the information from Equation 5, Bob will examine the questions Q and provide his own answers $\tilde{A}$, which may of course differ from the correct answers. The decryption software will then search through subsets $\tilde{A}_i \in \tilde{A}^*$, for each subset computing:

$$\tilde{K} = \mathbf{H}^{2^s}\left(O_i \oplus \bigoplus_{a_j \in \tilde{A}_i} \mathbf{H}\left(a_j || j\right)\right) \tag{6}$$

If Bob's subset of answers is correct, that is, $\tilde{A}_i = A_i$, then this decryption will successfully invert the decryption of Equations 4 and 3, and Bob will have recovered K_{M}. Any incorrect answer within $\tilde{A}_i$ will produce a pseudo-random $\tilde{K} \neq K_{\mathrm{M}}$.

Superficially, the encryption software must perform a search over all sets in $\tilde{A}^*$, as no information is provided about which answer was specified incorrectly. We hope that, in practice, there will be a manageable number of such subsets. A very helpful interface feature, however, is to provide a "don't know" answer for Bob, so that the decryption software can avoid searching over subsets known to contain a wrong answer.

4.2 Difficulty of an attack

The attacker receives the set of questions Q and must go about trying to guess enough of the answers to reconstruct the key. We assume that for each question, the sender can estimate a reasonable lower bound on the difficulty of an adversary guessing the correct answer. Specifically, we estimate the *min-entropy* H_∞ of each question, which is equal to $-\lg(p_{\max})$, where $p_{\max}$ is an attacker's estimated probability of the most likely answer.[1] We denote H^i_∞ for the min-entropy of question q_i.

Given our use of secret-sharing subsets, the attacker doesn't need to guess all answers successfully, only the answers in one subset $A' \in A^*$. There are many such subsets which will work, but the attacker can be assumed to target the weakest subset, that is, the subset A_{attack} which minimises:

$$H_{\mathrm{attack}} = \sum_{a_i \in A_{\mathrm{attack}}} H^i_\infty \tag{7}$$

The metric H_{attack} can be thought of as modelling an imaginary attacker who has an infinite supply of different messages sent from Alice to Bob, and will only guess the most likely answers to the weakest sufficient subset of answers to each message before discarding it and attacking the next message. This is the strongest possible attacker; an attacker which proceeds to guess less likely answers or larger subsets can only have lower guessing efficiency. The imaginary attacker with an infinite supply of messages will only succeed with probability $2^{-H_{\mathrm{attack}}}$ with one guess each, thus H_{attack} is an effective lower bound on the workload of an attacker.

Due to our key-strengthening, the attacker must actually perform an 2^s invocations of $\mathbf{H}$ to check each guess. Thus, the effective attacker workload is $H_{\mathrm{attack}} + s$ bits.

[1] Despite its common use for the purpose, Shannon entropy is not a sound measure of guessing difficulty. Min-entropy is an effective lower-bound on the difficulty of guessing a sample drawn from a known probability distribution.

4.3 Probability of successful decryption

We also consider the probability that Bob will be able to successfully decrypt the message. We assume that for each answer a_i, Bob has an associated probability of correctly recalling it, denoted as r_i. Although there are some known techniques to increase r_i, such as normalisation of answers to prevent mistakes due to spelling or punctuation [2,3], there is little recourse if Bob has legitimately forgotten too many answers. We assume that Bob will only supply answers once, he will not perform his own meta-search through likely possible answers, as this is unlikely to be acceptable from a usability standpoint.

Similar to the case for the attacker, Bob needs to remember some subset of answers $A' \in A^*$. The probability of doing so is:

$$p_{\text{success}} = \sum_{A' \in A^*} [\prod_{a_j \in A_i} r_j \cdot \prod_{a_k \notin A_i} (1 - r_k)] \tag{8}$$

4.4 Optimisation of parameters

The encryption software is free to choose the subsets A^* which allow for successful decryption given the user's choices for Q, and estimates of the min-entropy H^i_∞ recall probability r_i for each question. It can optimise either for maximum decryption probability given a required security parameter $H_{\min}$, or for maximum security given a required minimum decryption probability $p_{\min}$. In either case, we initialise our state as:

$$A^* = \{A\};\ H_{\text{attack}} = \sum_{a_i \in A} H_i;\ p_{\text{success}} = \prod_{a_j \in A} r_j \tag{9}$$

This corresponds to requiring that every answer is provided correctly, and gives maximal security and minimal decryption probability. We then add the subset $A' \notin A^*$ for which H_{attack}, as defined in Equation 7, is maximal, which becomes the new estimated strength of the entire encryption. After we add A', we update:

$$H_{\text{attack}} := \sum_{a_i \in A'} H^i_\infty; \quad p_{\text{success}} += \prod_{a_j \in A'} r_j \cdot \prod_{a_k \notin A'} (1 - r_k) \tag{10}$$

Note that, as we add to A^*, H_{attack} is monotonically decreasing while p_{success} is monotonically increasing, building up to their correct values as defined in Equations 7 and 8. The stopping conditions are obvious, either we continue until the next subset A' to add would result in $H_{\text{attack}} < H_{\min}$ if we have a minimum security requirement, or we continue until $p_{\text{success}} \geq p_{\min}$ if we have a minimum decryption probability requirement.

We can further reduce storage requirements by omitting any answer subset A' which is a strict superset of another valid answer subset A'', as the recipient can always use the smaller set A'' to decrypt if A' would have been possible. We call the reduced set of answer subsets $\hat{A}^*$.

4.5 Example values

We consider an example scenario in which Alice provides 20 questions, each of which has min-entropy $H_\infty = 4$ bits and recall probability $r = 0.9$. Of course, our system is designed to support different values for each question, but we consider a simple example to calculate the resulting message size and security. A reasonable choice is to enable decryption if at least 16 answers are guessed correctly:

$$A^* = \{A' \in A : |A'| \geq 16\} \tag{11}$$

The total number of correctly decrypting sets is then:

$$|A^*| = \binom{20}{20} + \binom{20}{19} + \binom{20}{18} + \binom{20}{17} + \binom{20}{16} = 6,196 \tag{12}$$

As discussed in Section 4.4, we can discard many of these subsets which are supersets of others, resulting in an optimal set of size:

$$|\hat{A}^*| = \binom{20}{16} = 4,485 \tag{13}$$

The probability of successful decryption, as defined in Equation 8, is:

$$p_{\text{success}} = \sum_{A' \in A^*} [\prod_{a_j \in A_i} r_j \cdot \prod_{a_k \notin A_i} (1 - r_k)] = \sum_{16 \leq i \leq 20} 0.9^i \cdot 0.1^{20-i} = 0.957 \tag{14}$$

The attacker's workload, defined in Equation 7, is that of any set of 16 answers, which is:

$$H_{\text{attack}} = \sum_{a_i \in A_{\text{attack}}} H_\infty^i = 16 \cdot 4 = 64 \tag{15}$$

Assuming we desire 80 bits of security, we require each offset O_i to be 80 bits. Each offset represents a subset of at least 16 of the 20 answers, which we can efficiently encode by listing the excluded items in a maximum of 20 bits (5 bits each for up to 4 excluded answers). Thus, we need to send a maximum of $4,485 \cdot 100 = 448,500$ bits to represent O, which is ≈ 55 kB.

Our attacker will have a maximal success probability of 2^{-64}, so we can use $s = 16$ to make her total workload equivalent to 2^{80} invocations of the hash function $\mathbf{H}$.

Finally, even if our intended recipient must search every subset in A^*, this will require a total of $6,196 \cdot 2^{16} \approx 2^{28.1}$ invocations of $\mathbf{H}$.

5 Sample experiment

The author conducted a small experiment to test the feasibility of the proposed protocol. Questions were written for eight different people whom the author has

known for an extended period of time. One hour was spent writing questions and estimating their strength and probabilities for each recipient.[2] The resulting number of questions and message sizes are listed in Table 1.

Each recipient received 12–16 questions, and needed to answer between 5 and 12 of them in order to successfully decrypt. This variation was due to the varying estimates of guessability between different recipients. For some it was easier to craft difficult-to-guess questions and for others it was necessary to rely on a larger number of relatively weak questions. This also led to a large variation in message size from 1–70 kB, depending mostly on the number of sufficient subsets for decryption which required sending additional offsets.

Relation	$\|Q\|$	$H_{\max}$	p_{success}	$\|A^*\|$	$\|\hat{A}^*\|$	$\min(\|A_i\|)$	storage (kb)
ex-partner	15	76	0.724	189	128	12	1
sister	13	87	0.878	597	284	8	3
mother	12	88	0.937	660	312	7	3
brother	16	98	0.979	9352	2240	8	40
former roommate	13	93	0.974	9352	2240	5	40
partner	16	89	0.977	9819	2600	8	45
father	14	95	0.959	10032	2679	6	47
childhood friend	16	101	0.989	13696	3776	8	70

Table 1. Number of questions sent and total message overhead for each of the 8 study participants, requiring a minimum estimated security of $H_{\text{attack}} = 64$ in each case.

Each recipient attempted to answer their own questions only once. Every participant answered enough questions correctly to successfully decrypt, though every recipient was incorrect on multiple questions. The totals are listed in Table 2. Overall, 82 of 114 questions, or 72%, were answered correctly. By the author's estimates during the experiment, close to 85% of questions were expected to be answered correctly. Breaking down the caused of incorrect answers, 6% were due to spelling disagreement, and a further 7% were due to synonym replacement.[3] 16% of questions were legitimately forgotten or not known, close the original estimate of success. This suggests a bias towards ignoring the possibility of input mistakes when estimating the probability of a correct answer.

Each recipient also was encouraged to attempt to guess the answers to every other recipients' questions as many times as they were willing to. Many of the recipients, being family members and long-term friends, knew each other well. Thus there were a number of successful guesses, also listed in Table 2. A total of 13 questions, over 10%, were successfully guessed by another participant in

[2] The one-hour time limit seems to be a lower limit. It is surprisingly difficult to remember and write good questions for use in this system.

[3] For example, one recipient answered "Pt. Reyes" instead of "Drake's Bay" which are equivalent names for the same physical place.

the study. Nearly all of these were either one of the author's siblings guessing a question intended for the other sibling, or one of the author's parents guessing a question intended for the other parent, demonstrating the difficulty of separating these pairs of people. None of the recipients had enough questions guessed to successfully decrypt, although in several cases enough questions were asked to push H_{attack} below 40 bits (prior to key-strengthening).

recipient	$\lvert Q\rvert$	$\lvert A^*\rvert$	successful answers	guessed answers
ex-sig. other	15	12	12	0
sister	13	8	10	2
mother	12	7	8	3
brother	16	8	13	4
former roommate	13	5	10	1
sig. other	16	8	14	0
father	14	6	7	3
childhood friend	16	8	10	0

Table 2. Number of questions successfully answered by intended recipients, as well as guessed correctly by any other recipient.

Overall, the experiment suggested that such a protocol is possible with reasonable real-world parameters (one hour of time to create questions and less than 100 kB of storage overhead). All messages were successfully decrypted with only very rudimentary software making no effort to correct typos or spelling mistakes. However, the experiment also demonstrated that accurately estimating the probability of recalling answers is very difficult, and that acquaintance attacks by people who know the sender or recipient can be a significant problem.

6 Open questions

6.1 Sender authentication

So far we have only considered the question of confidentiality of the message being sent to Alice, and this was all that was needed in prior systems which only considered encryption to oneself. In our scenario, we may ask if there is any secure way for Bob to ensure that a message he receives encrypted in our scheme really came from Alice, as claimed. We conjecture that this problem is far more difficult to completely solve. There does exist a degree of implicit authentication in that Bob knows the sender is somebody who knew a significant amount of personal information. In our proposal, we have included a MAC of all correct answers in our system as a weak indication to the recipient that the sender did in fact know all of the correct answers. We could consider adding some additional personal information to the encryption which was not already used.

We feel that this provides little security against a malicious attempt to forge a message from Alice. Presumably, such an attack could be constructed by, for

example, breaking into Alice's email address and reading old correspondence, burgling Alice's home, or dumpster diving to collect discarded material. This sort of attack is far easier to pull off than a compromise of the confidentiality of the system, because the attacker must only find enough information that seems "personal enough" for Bob to assume it was Alice herself which sent the message. To decrypt an intercepted message, the attacker must find specifically the information chosen by Alice to encrypt, which is far more difficult.

6.2 Fuzzy matching

Another desirable trait is fuzzy matching of answers. That is, instead of requiring that Bob answer some subset of questions *exactly*, we can allow him to answer fewer questions exactly and be close on some questions. For example, if the question is, "When is cousin Jeff's birthday?", Bob may guess April 12 when the correct date is April 14. Intuitively, Bob has demonstrated some knowledge of the correct answer while still being wrong. An ideal system would enable the recipient to learn a varying amount of the pre-key based on closeness to a correct answer. This may be useful in textual questions as well, if a name is misspelled but still largely correct.

In our current system, this functionality is missing. The closest we can do is normalise answers, say, by removing spaces and capitalisation, or more aggressively by applying the Soundex algorithm to mask spelling errors. In the case of numbers or dates, they can be rounded. Neither of these methods is ideal, as they assist with guessing attacks, and may not be helpful in that two close answers may still be rounded to different values. An open question is how to design a system that will allow for fuzzy matching of answers.

References

1. Ellison, C., Hall, C., Milbert, R., Schneier, B.: Protecting secret keys with personal entropy. Future Gener. Comput. Syst. **16** (2000) 311–318
2. Frykholm, N., Juels, A.: Error-tolerant password recovery. In: CCS '01: Proceedings of the 8th ACM conference on Computer and Communications Security, New York, NY, USA, ACM (2001) 1–9
3. Just, M.: Designing and evaluating challenge-question systems. IEEE Security & Privacy (2004)
4. Rabkin, A.: Personal knowledge questions for fallback authentication: Security questions in the era of facebook. SOUPS: Symposium on Usable Privacy and Security (2006)
5. Griffith, V., Jakobsson, M.: Messin' with texas: Deriving mother's maiden names using public records. Applied Cryptography and Network Security (2005)
6. Xu, W., Zhou, X., Li, L.: Inferring privacy information via social relations. International Conference on Data Engineering (2008)
7. Lindamood, J., Kantarcioglu, M.: Inferring private information using social network data. Technical Report UTDCS-21-08, University of Texas at Dallas Computer Science Department (2008)

8. Jakobsson, M., Stolterman, E., Wetzel, S., Yang, L.: Love and authentication. In: CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems, New York, NY, USA, ACM (2008) 197–200
9. Jakobsson, M., Yang, L., Wetzel, S.: Quantifying the security of preference-based authentication. In: DIM '08: Proceedings of the 4th ACM workshop on Digital identity management, New York, NY, USA, ACM (2008) 61–70
10. Kelsey, J., Schneier, B., Hall, C., Wagner, D.: Secure applications of low-entropy keys. In: ISW '97: Proceedings of the First International Workshop on Information Security, London, UK, Springer-Verlag (1998) 121–134