Implementing Public Key Cryptography with Keyword Search

Prashanth Ramakrishna
pmr347@nyu.edu

Dov Salomon
dms833@nyu.edu

Hao Shu
hs3812@nyu.edu

Jonathan Alter
ja3943@nyu.edu

May 12, 2020

1 Introduction

For Project 2, we decided to choose Option 6. Option 6 requires summarizing a paper, implementing it's constructions, then testing the performance of our implementations as relevant parameters grow. The paper we chose to implement is entitled, "Public Key Cryptography with Key Word Search". This paper attempts to solve the following question:

Alice is an email user who would like to access her email on multiple devices. However, she would like emails to be sorted, via routing, to different devices based on certain keywords they contain. For example, suppose Bob sends Alice an email containing the keyword "urgent". This email should be routed to Alice's pager rather than her desktop. Such a scheme would be trivial to implement, of course, if Bob's email were in plaintext. The gateway, given a set of keywords, could simply search the plaintext and act accordingly. But, how can such a scheme be accomplished if Bob's email is encrypted and, along with it, the keywords contained therein?

In this case, as should be apparent, routing decisions cannot be made by the gateway because keywords are hidden. This paper develops a scheme whereby Alice enables the gateway to test whether a specified set of keywords is contained in encrypted emails being received without learning anything else. Formally, Bob sends Alice an encrypted email of the following form:

$$E_{A_{pub}}(M) \;||\; \texttt{PEKS}(A_{pub}, W_1) \;||\; \ldots \;||\; \texttt{PEKS}(A_{pub}, W_m) \tag{1}$$

where A_{pub} is Alice's public key, $E_{A_{pub}}$ refers to some standard public key encryption using Alice's public key, PEKS is a *Public-Key Encryption with Keyword Search* of each keyword $W_1, \ldots, W_m$ sent by Bob in message M. The advantage of this form is that Alice can provide a trapdoor T_W which will allow the gateway to test whether any of the keywords associated with Bob's message match Alice's chosen keyword W. That is, given $\texttt{PEKS}(A_{pub}, W')$ and T_W, the gateway has the ability to determine if $W = W'$. Note that if $W \neq W'$, then the gateway learns nothing further about the email other than that $W \notin M$.

Definition 1.1. *Non-Interactive PEKS*

1. $KeyGen(s)$: $KeyGen$ takes the key length security parameter s as input and return a public-private key pair, denoted $\{A_{pub}, A_{priv}\}$
2. $\texttt{PEKS}(A_{pub}, W)$: Given public key A_{pub} and word W, PEKS returns the searchable encryption of W.
3. $T(A_{priv}, W)$: Alice uses her private key A_{priv} to create a trapdoor T_W for chosen word W.
4. $Test(A_{pub}, S, T_W)$: Given Alice's public key, searchable encryption of W', $S = \texttt{PEKS}(A_{pub}, W')$, and trapdoor $T_W = T(A_{priv}, W)$, $Test$ returns boolean $W = W'$.

First, Alice uses $KeyGen$ to produce her public/private key pair. Then, she uses T to produce a trapdoor T_{W_i} for each $W_i \in W^*$, where W^* is the set of chosen keywords $\{W_1, \ldots, W_m\}$. This set of trapdoors is denoted T_{W^*} and is given by Alice to her mail server. Alice When Alice's mail server receives a message from Bob

$$E_{A_{pub}}(M) \;||\; S_1 \;||\; \ldots \;||\; S_m$$

it uses $Test$ to determine whether message contains any member of W^*.

Security for a PEKS scheme is defined against active attacker $\mathcal{A}$ who, given parameter s, is able to query an oracle for the trapdoor T_W of any chosen word $W \in \{0,1\}^*$. Despite this ability, $\mathcal{A}$ should be unable to distinguish $\texttt{PEKS}(A_{pub}, W_1)$ and $\texttt{PEKS}(A_{pub}, W_2$ if he has T_{W_0} but not T_{W_1}. The challenge for the attacker is, when given two new keywords W_0, W_1 for which he doesn't have the trapdoor, and searchable encryption $\texttt{PEKS}(A_{pub}, W_b$, where $b \in \{0,1\}$, to guess b. $\mathcal{A}$ wins the security game if his advantage,

$$Adv_{\mathcal{A}}(s) = |P(b = b') - \frac{1}{2}| \tag{2}$$

where b' is $\mathcal{A}$'s guess, is non-negligible.

Remark 1.1. The general construction given in Def 1.1 is not secure against a chosen cipher text attack, since $\mathcal{A}$ can simply decrypt after reorder the searchable keyword encryptions appended to M. However, Def 1.1 does ensure semantic security for the message form given in Eq 1 so long a $E_{A_{pub}}$ is itself semantically secure. With modifications, chosen cipher-text security can be achieved. Indeed, in the paper, a chosen ciphertext secure IBE system, IND-ID-CCA, is detailed.

The paper gives two different constructions of the general PEKS scheme. The first construction is based on a bilinear function and the second, simpler albeit less efficient one, is based only on general trapdoor permutations. They are detailed in Section 2 and Section 3. In this project, both constructions are implemented

2 Bilinear Construction

This construction makes depends on the hardness of a cousin to the Computational Diffie-Hellman problem known as the **Bilinear Diffie Hellman Problem**.

Definition 2.1. *Bilinear Diffie Hellman Problem (BDH)* Let G_1, G_2 be two groups of fixed prime order p with generator sets g_1 and g_2, respectively. Further, let $e : G_1 \times G_1 \to G_2$ be a bilinear function. That is, e satisfies the following properties:

1. Computability: $\forall x_1, x_2 \in G_1$, $\exists$ polytime algorithm A to compute $e(x_1, x_2) \in G_2$
2. Bilinearity: $\forall x_1, x_2 \in [1, p]$ and $\forall g \in G_1$, $e(g^{x_1}, g^{x_2}) = e(g, g)^{x_1 x_2}$.
3. Non-degeneracy: $\forall g \in g_1$, $e(g, g) \in g_2$.

Now, fix $g \in g_1$. Given $g, g^a, g^b, g^c \in G_1$, compute $e(g, g)^{abc} \in G_2$. If all polytime algorithms have a negligable advantage in solving BDH, then we say BDH is intractable.

The PEKS bilinear map construction, in accordance with Def requires G_1, G_2, and e. In addition, however, it requires two hash functions $H_1 : \{0,1\}^* \to G_1$ and $H_2 : G_2 \to \{0,1\}^{log(p)}$. The scheme, then consists of the four functions defined in Def 1.1.

1. $KeyGen$: Security parameter p is the prime order of G_1 and G_2. $KeyGen$ randomly chooses $\alpha \in Z_p^*$ and $g \in g_1$. $A_{pub} = [g, h = g^\alpha]$ and $A_{priv} = \alpha$. Finally, $\{A_{pub}, A_{priv}\}$ is returned.
2. $\texttt{PEKS}(A_{pub}, W)$: First, $r \in Z_p^*$ is randomly chosen. Then, $t = e(H_1(W), h^r) \in G_2$ is computed. Finally, $[g^r, H_2(t)]$ is returned.
3. $T(A_{priv}, W)$: given chosen keyword W, T returned trapdoor $T_W = H_1(W)^\alpha \in G_1$.
4. $Test(A_{pub}, S, T_W)$: Let $S = [A, B]$. $Test$ returns the boolean $H_2(e(T_W, A)) = B$.

The mechanics of the encryption scheme – how and when these functions are used – are inherited from the generic construction given in Def 1.1. About this scheme, we have our first theorem.

Theorem 1. *The non-interactive searchable encryption scheme (PEKS) with the bilinear map construction above is semantically secure against a chosen keyword attack in the random oracle model assuming BDH is intractable.*

The proof of Thm 1 is too extensive to include here. The proof's high level approach, however is not. We have attack algorithm $\mathcal{A}$ with advantage ϵ against PEKS. $\mathcal{A}$ makes $\leq q_{H_2}$ queries to H_2 and $\leq q_T$ trapdoor queries. Further, we have algorithm $\mathcal{B}$ that solves BDH with probability $\epsilon' = \frac{\epsilon}{e q_T q_{H_2}}$, where e is the base of the natural log. Because, by construction, $runtime(\mathcal{A}) \approx runtime(\mathcal{B})$, the BDH assumption holding for G_1, ϵ' will be negligible, and thus, so will ϵ. The proof, therefore, requires the construction of $\mathcal{B}$ such that ϵ' is negligible.

3 Trapdoor Construction

This second PEKS construction makes use of general trapdoor permutations. It relies on two assumptions:

1. $|W^*| \leq f(s)$, where W^* is the set of user-chosen keywords and f is a polynomial function that takes security parameter s as input. That is, the number of user-chosen keywords is polynomially bounded in s.
2. Let $\Sigma \subset \{0,1\}^*$ be the set of all words in the dictionary $\{0,1\}^*$. Then, $|\Sigma| \leq f(s)$, where f is a polynomial function that takes security parameter s as input. That is, the total number of words in the dictionary is polynomially bounded in s.

Further, we aim for our construction to be **source indistinguishable**.

Definition 3.1. *Source Indistinguishability* Suppose key generation algorithm G, encryption algorithm E, and decryption algorithm D form the encryption scheme (G, E, D). Next, suppose we have attacker $\mathcal{A}$, challenger $\mathcal{B}$, and security parameter s. Source indistinguishability is defined by the following security game in which both attacker and challenger are given s.

1. $\mathcal{B}$ runs $G(s)$ twice to obtain two public/private key pairs $\{Pub_0, Priv_0\}$ and $\{Pub_1, Priv_1\}$.
2. $\mathcal{B}$ randomly chooses $M \in \{0,1\}^s$ and $b \in \{0,1\}$. Then $\mathcal{B}$ computes $C = E_{Pub_b}(M)$ and gives (M, C) to $\mathcal{A}$.
3. $\mathcal{A}$ returns his guess b', winning if $b = b'$.

That is, $\mathcal{A}$ wins if he is able to correctly guess which public key was used to encrypt the message given by the challenger with non-negligible advantage, defined as

$$AdvSI_{\mathcal{A}}(s) = |P(b = b') - \frac{1}{2}| \tag{3}$$

We call an encryption scheme (G, E, D) source indistinguishable if $\mathcal{A}$'s advantage in the security game defined above is negligible.

Now, let (G, E, D) be a source indistinguishable public-key crypto system and s be its security parameter. The four functions given in 1.1 are defined for this PEKS trapdoor permutation scheme:

1. $KeyGen$: $\forall W \in \Sigma$, use $G(s)$ to produce public/private key pair $\{Pub_W, Priv_W\}$. The PEKS public and private keys are sets $A_{pub} = \{Pub_W | W \in \Sigma\}$ and $A_{priv} = \{Priv_W | W \in \Sigma\}$ respectively.
2. $\texttt{PEKS}(A_{pub}, W)$: First, $M \in \{0,1\}^s$ is randomly chosen. Then, message M is encrypted using Pub_W and $\texttt{PEKS}(A_{pub}, W) = (M, E(Pub_W, M))$ is returned.
3. $T(A_{priv}, W)$: Trapdoor $T_W = Priv_W$ is returned.
4. $Test(A_{pub}, S, T_W)$: $Test$ returns boolean $D(T_W, S) = 0^s$.

Note that Assumption 2 must hold in order for keys to be polynomial size in s. About this scheme, then, we have our second theorem.

Theorem 2. The PEKS trapdoor permutation scheme given above is semantically secure assuming the underlying public key encryption scheme (G, E, D) is source-indistinguishable.

The proof of Theorem 2 is only sketched in the paper. Supposing an attacker $\mathcal{A}$, with advantage $Adv_{\mathcal{A}}(s) > \epsilon(s)$, the proof relies on the construction of an additional attacker $\mathcal{B}$ who is able to break the source-indistinguishability of (G, E, D) with advantage $AdvSI_{\mathcal{B}} > \epsilon(s)/k^2$, where $|\Sigma| = k$.

4 Testing

Our implementations of the above two constructions are contained in files **bilinear.py** and **TrapdoorPermutation.py**, respectively. We conducted tests for both constructions, exploring how $|W^*|$ and s – interpreted as group order in the bilinear construction and key size in the trapdoor construction – affects running time. The results are shown in Fig. 1.

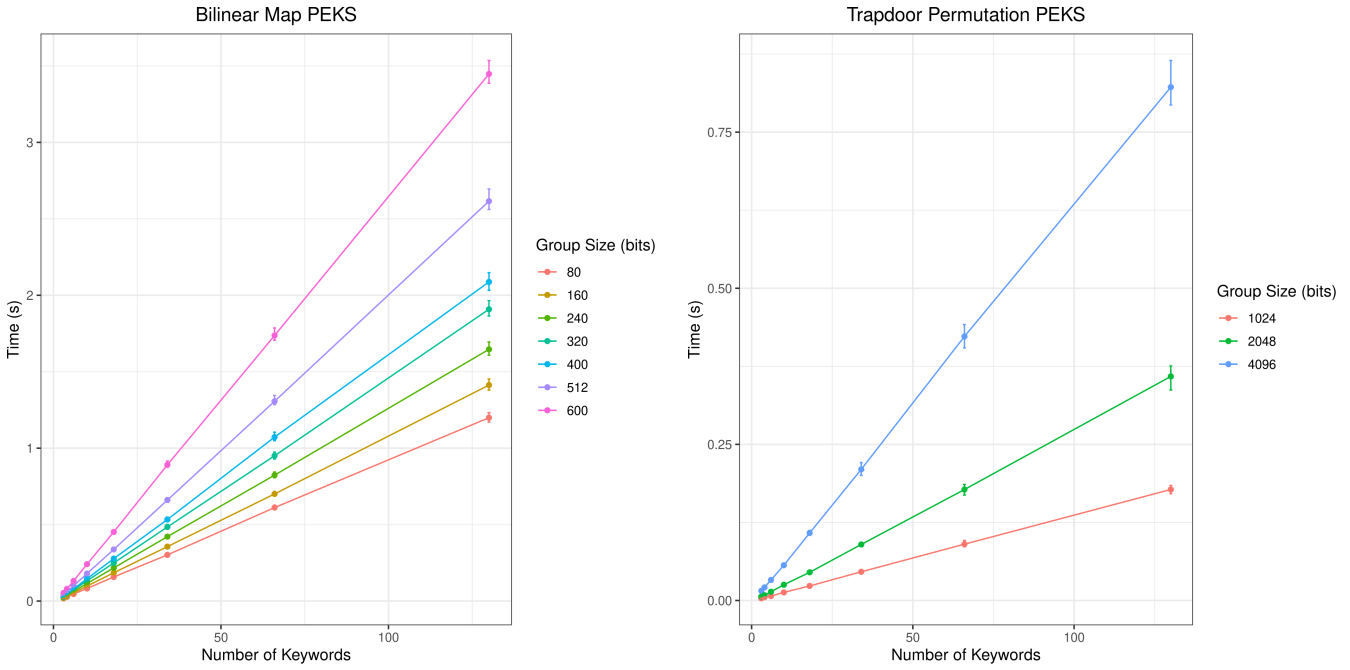

Figure 1: *This plot shows the running time of both the Bilinear Map* `PEKS` *(panel 1) and Trapdoor Permutation* `PEKS` *(panel 2) function against security parameter s and number of keywords $|W^*|$. 95% confidence intervals are shown and drawn from test batches of size 50.*

We see from Fig. 1 that the Bilinear Map `PEKS` takes longer, even for small security parameters, than the Trapdoor Permutation `PEKS`. This is expected, though, since permutation is less expensive than exponentiation. Additionally, we need to take into consideration the relative performance of the libraries that we are using to implement each construction. Note from Fig. 2 the speed per word increases is qualitatively exponential in the Blinear Map construction and linear in the Trapdoor Permutation construction. This is expected given the exponentiation of bilinear function e and the nature of permutations. For more definitive relationships, however, tests should be run with a greater number of security parameter choices. The inefficiency of the Trapdoor Permutation construction is expressed in its $KeyGen$ function. $KeyGen$ is roughly constant given the Bilinear Map construction as the number of keywords increases. It does not, however given the Trapdoor Permutation construction. This is shown in Fig. 3. Fig. 4 shows the running times for the $Test$ function given both constructions. $Test$ proves to be relatively inexpensive. We recognize that the greatest amount of time given the Bilinear Map construction is spent on encryption, whereas the greatest amount of time given the Trapdoor Permutation construction is spent on key generation. Moreover, key generation given the Trapdoor Permutation construction is far more expensive than encryption given the Bilinear Map construction. In sum, then, as noted in the paper, the Bilinear Map construction is more efficient than the Trapdoor Permutation construction. Testing was optimized via parallelization.

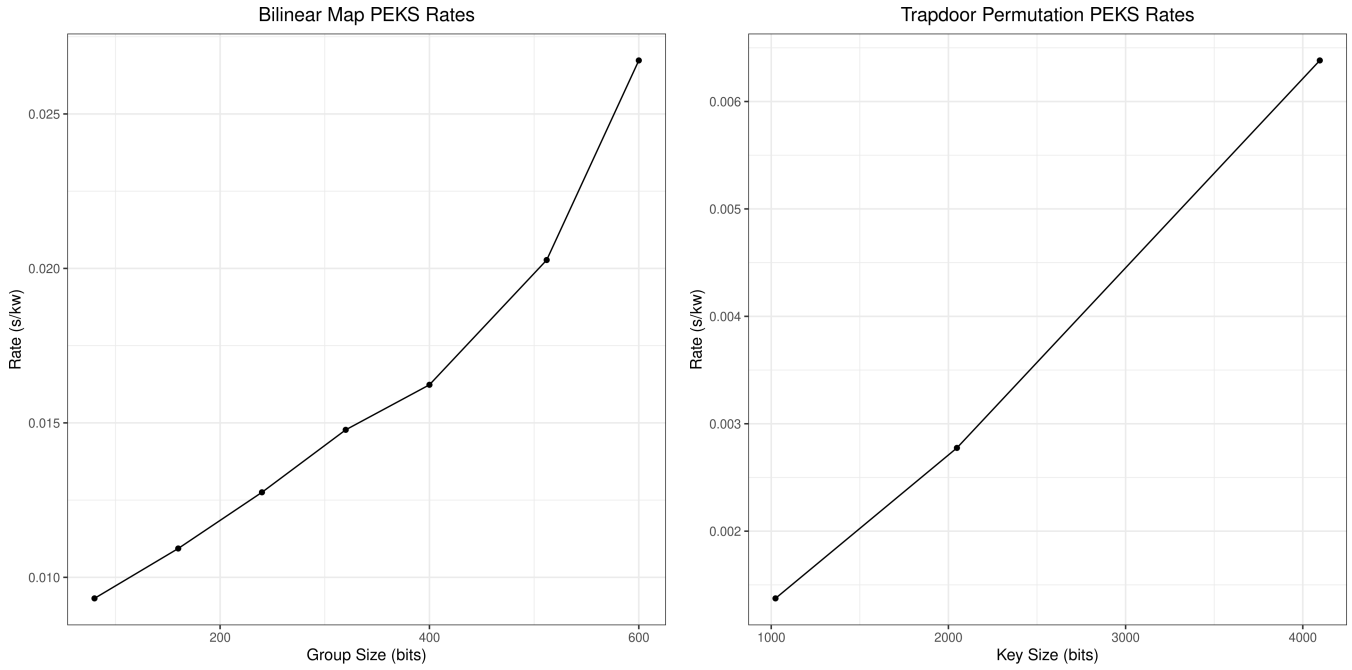

Figure 2: *This plot shows rate of average runtime increase (s) per keyword for both the Bilinear Map* `PEKS` *(panel 1) and Trapdoor Permutation* `PEKS` *(panel 2) functions as against security parameter s.*

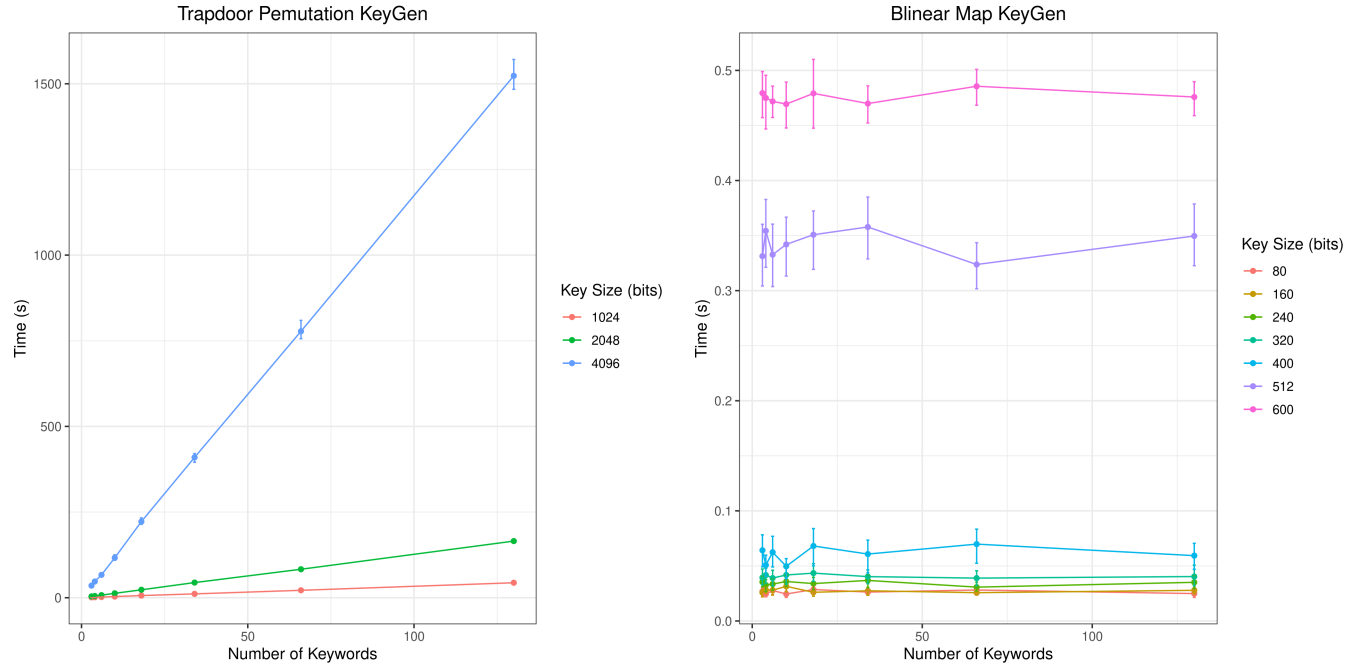

Figure 3: *This plot shows the running time of KeyGen given the Trapdoor Permutation* `PEKS` *constructions as a function of security parameter s and number of keywords* $|W^*|$*. 95% confidence intervals are shown and drawn from test batches of size 50.*

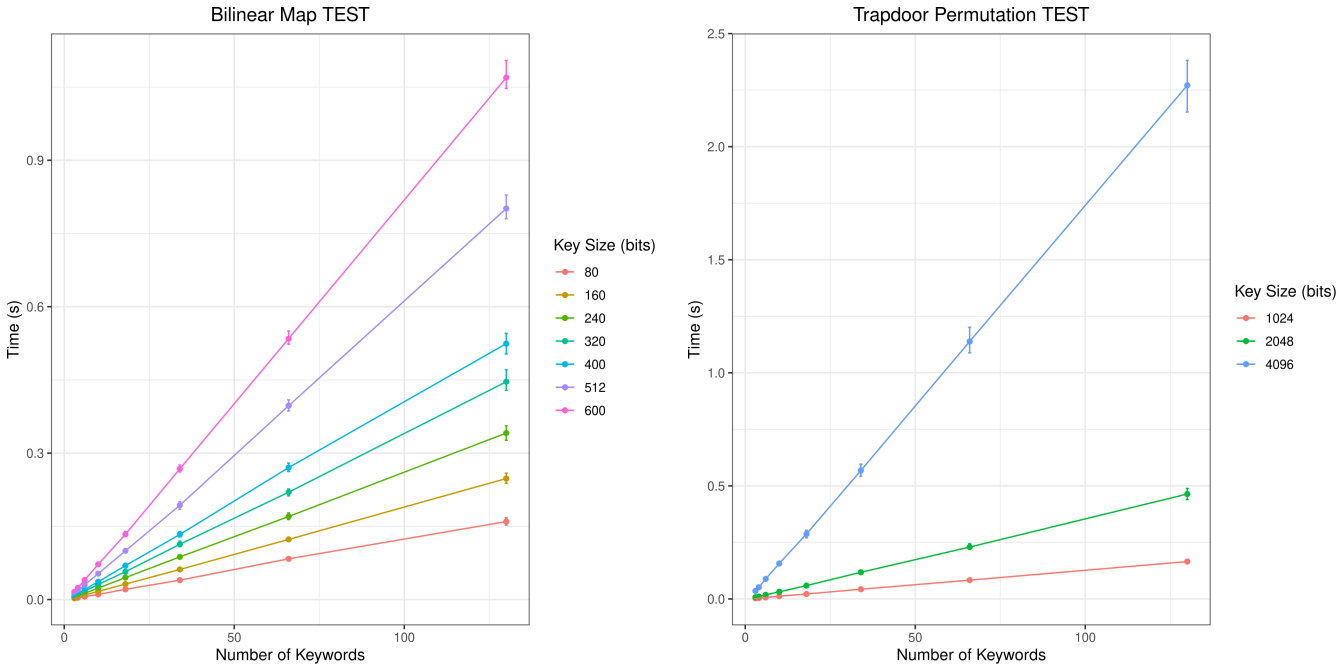

Figure 4: *This plot shows the running time of the Test function given the Blinear Map (panel 1) and Trapdoor Permutation* `PEKS` *(panel 2) constructions against security parameter s and number of keywords* $|W^*|$. 95% *confidence intervals are shown and drawn from test batches of size 50.*

5 Conclusion/Acknowledgments

We would like to thank the authors of the paper we reviewed – Dan Boneh, Rafail Ostrovosky, Giuseppe Persiano, and, likely by design, our professor Giovanni Di Crescenzo.