Cryptographic treatment of CryptDB's Adjustable Join

Raluca Ada Popa and Nickolai Zeldovich
MIT CSAIL

March 25, 2012

1 Introduction

In this document, we provide a cryptographic treatment of the adjustable join protocol from CryptDB [5]. We also discuss how our scheme could be used outside of CryptDB because it provides a simple functionality that may be needed in other settings. Intuitively, it is a pseudorandom permutation where an external party not knowing the secret key can nonetheless adjust a ciphertext under one key to a ciphertext under a different key, given an adjustment token from a party that knows the secret key.

We assume the reader is familiar with basic cryptographic and elliptic curve notions (e.g., computational indistinguishability, elliptic curve groups, pseudorandom permutation ensembles). These notions are explained in [2], [3] and [4].

1.1 CryptDB's setting

The problem setup for the join in CryptDB consists of a trusted proxy and an untrusted server as shown in Figure 1. The server stores many columns each containing many data items. The proxy would like to enable the server to compute a *join* between certain pairs of columns; to perform a join between two columns means to identify all pairs of items, one from each column, which are equal to each other.

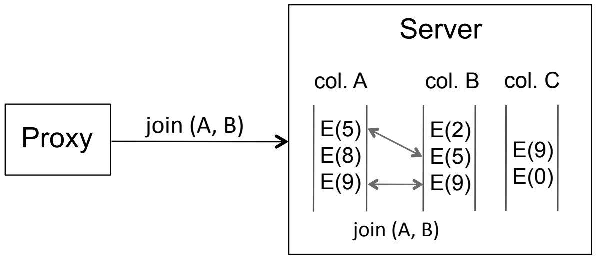

Figure 1: Problem setup. E denotes encryption.

The practical setting of CryptDB and its security goals place the following constraints on a join solution:

- The proxy should be able to enable the server to join certain columns, but the server should not learn anything about the items' values other than the join relations, or about join relations for columns which the proxy did not wish to join. For example, if the proxy wants to join columns A and B and to join C and D, the server should learn join relations between A and B, as well as join relations between C and D, but it should not learn join relations between A and C, A and D, B and C, or B and D. However, we allow join transitivity. That is, if the proxy wants to join B and E, we allow the server to see join relations among A and E.

- The server should not be able to infer join relations between two columns, A and B, before the proxy actually asks the server to do so.
- The solution should not require the proxy to know a priori (at the time when the data items are stored at the server) the columns that will be joined.
- The server should be able to determine join relations between two columns by simply performing equality checks. This constraint is for performance and deployability: the server can now use the equality operator and indexes built in database systems to process joins, and CryptDB does not need to change the internals of database systems.
- The proxy should not perform re-encryption of the data.

2 Preliminaries

2.1 Notation

DPT stands for deterministic polynomial-time algorithm and PPT stands for probabilistic polynomial-time algorithm.

If $\mathcal{D}$ is a distribution, $x \leftarrow \mathcal{D}$ means that x is a sample from $\mathcal{D}$. If $\mathcal{D}$ is a finite set, $x \leftarrow \mathcal{D}$ denotes drawing x uniformly at random from the items in $\mathcal{D}$. If A is a PPT, $x \leftarrow A(y)$ denotes drawing a sample x from the output distribution generated by running A on input y.

Let $\mathsf{negl}(k)$ be a negligible function of k (i.e., smaller than one over any polynomial in k for sufficiently large k).

We define an *enumerable space* $\mathcal{D}$ to be an ordered space of values for which there exists a DPT algorithm that takes as input a value $v \in \mathcal{D}$ and returns $v + 1$, the next value after v in order, if v is not the last value, and for which there exists an initial value 0, starting from which all elements of $\mathcal{D}$ can be enumerated using the $v + 1$ algorithm.

2.2 Elliptic curves (EC) and the Decision Diffie-Hellman problem

We first present some intuition in elliptic curves for the unfamiliar reader, but please refer to [3] as a starter for a formal treatment. An elliptic curve E over a finite field $\mathbb{F}_q$ is usually a set of points in two-dimensional space together with one point "at infinity" that form a group with respect to addition of points. To add two points, one draws a line between the two points and takes the inverse (intuitively, the reflection) of the point at the intersection of this line and the elliptic curve, with some special treatment given to the infinity point. Multiplication of a point P by a scalar u works by performing repeated additions, using a $O(\log u)$ algorithm.

In terms of hardness assumptions, multiplication of a point P by a scalar u in some EC groups has similar hardness properties as exponentiation of a group element in some classical non-EC groups. For example, in some classical group $\mathbb{G}$, the discrete logarithm is considered hard, meaning that given $g \in \mathbb{G}$ and $h = g^u \in \mathbb{G}$, it is hard to compute u. Similarly, the discrete logarithm is also believed to be hard in some elliptic groups: given $P \in E$ and $H = Pu \in E$, it is hard to compute u. CryptDB's paper [5] chooses to use the notation P^u instead of Pu to convey intuitively why computing u is hard; in this document, nevertheless, we will use the conventional notation Pu.

In the algorithms we present, we will follow the convention that variables with capital letters represent points on an elliptic curve and lowercase letters indicate scalars, whenever applicable.

Elliptic-Curve Decisional Diffie-Hellman (ECDDH). The security of our scheme relies on the ECDDH assumption, which is a standard assumption: NIST put forth a set of elliptic curves in which the ECDDH assumption can be considered true, and ECDDH has been used in a variety of previous protocols, such as [4]. Intuitively, the assumption states that, for E_P an elliptic curve group of order p, the following two distributions are computationally indistinguishable:

$$P_0 \leftarrow E_P, P_1 \leftarrow E_P, u_0 \leftarrow \mathbb{Z}_p, u_1 \leftarrow \mathbb{Z}_p \quad : (P_0, P_1, P_0 u_0, P_1 u_1) \tag{1}$$
$$P_0 \leftarrow E_P, P_1 \leftarrow E_P, u_0 \leftarrow \mathbb{Z}_p \quad : (P_0, P_1, P_0 u_0, P_1 u_0). \tag{2}$$

Basically, a polynomial adversary cannot tell if the last term is a random number (as in 1) or it uses the same scalar (u_0) as the third term (as in 2).

We now present the ECDDH assumption formally in a format most useful for our proofs.

Let CurveGen be an algorithm that, on input a security parameter k, outputs an elliptic curve's parameters:

- q for a finite field $\mathbb{F}_q$,
- elliptic curve coefficients in $\mathbb{F}_q$ that define an elliptic curve E over $\mathbb{F}_q$, and
- a curve point $P \in E$ of order p, where p is a positive prime integer, and $|p| = k$.

Let E_P denote the cyclic group of points on the elliptic curve E generated by the point P.

The ECDDH assumption is believed to hold for elliptic curves over prime-order finite fields, for some large prime, which also have large embedding degree. The choice of elliptic curves to use for an implementation specifies the CurveGen algorithm.

Definition 1 (ECDDH on elliptic curves generated by CurveGen)**.** *The ECDDH assumption holds for elliptic curves generated by* CurveGen*, if for all polynomial-size family of circuits* $\{\mathsf{Adv}_k\}_k$*,*

$$\Pr\left[\begin{array}{l}\mathsf{params} = (q, E, p, P) \leftarrow \mathsf{CurveGen}(1^k);\\ P_0, P_1 \leftarrow E_P;\\ u_0, u_1 \leftarrow \mathbb{Z}_p; b \leftarrow \{0,1\} :\\ \mathsf{Adv}_k(\mathsf{params}, P_0, P_1, P_0 u_0, P_1 u_b) = b\end{array}\right] < 1/2 + \mathsf{negl}(k).$$

3 Adjustable join definition

We define what an adjustable join scheme is and then what it means for such a scheme to be secure.

Definition 2 (**Adjustable join**)**.** *An adjustable join scheme is a tuple of algorithms* (KeyGen, Enc, Token, Adj) *over an enumerable domain space* $\mathcal{D} = \{\mathcal{D}_k\}_{k\in\mathbb{N}}$*, where* k *is the security parameter:*

- ***Key generation.*** *The PPT algorithm* KeyGen *takes a unary representation of the security parameter and outputs system parameters and a secret key:* $(\mathsf{params}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^k)$.
- ***Encryption.*** *The DPT algorithm* Enc *takes the system parameters, the secret key, a message* $m \in \mathcal{D}_k$*, and a column index* $i \in \mathcal{D}_k$*, and outputs a ciphertext* C*:* $C = \mathsf{Enc}(\mathsf{params}, \mathsf{sk}, m, i)$.
- ***Adjustment token.*** *The DPT algorithm* Token *takes the system parameters, the secret key, as well as two columns,* $i, j \in \mathcal{D}_k$ *, and outputs a token* t*:* $t = \mathsf{Token}(\mathsf{params}, \mathsf{sk}, i, j)$.
- ***Adjustment.*** *The DPT algorithm* Adj *takes the system parameters, a ciphertext* C*, and a token* t *and outputs a new ciphertext* C'*:* $C' = \mathsf{Adj}(\mathsf{params}, C, t)$.

We call our implementation of an adjustable join scheme (presented in Sec. 4) ADJ-JOIN. To avoid confusion, we mention that our paper [5] refers to JOIN-ADJ as the encryption part of ADJ-JOIN, which we call Enc here.

Definition 3 (Correctness). *An adjustable join scheme is correct, if for all* $(\mathsf{params}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^k)$*, for all* $m_0,\ m_1 \in \mathcal{D}_k$ *with* $m_0 \neq m_1$*, for all columns* $i,\ j \in \mathcal{D}_k$*, the following properties hold. Let* $C_b^l = \mathsf{Enc}(\mathsf{params}, \mathsf{sk}, m_b, l)$ *for* $b \in \{0,1\}$ *and* $l \in \{i,j\}$ *be the encryption of* m_b *for column* l*; let* $t_{i\to j} = \mathsf{Token}(\mathsf{params}, \mathsf{sk}, i, j)$ *be the adjustment token from column* i *to column* j*. Then,*

1. $\mathsf{Adj}(\mathsf{params}, C_0^i, t_{i\to j}) = C_0^j$*, and*
2. $C_0^i \neq C_1^i$.

Intuitively, the two conditions ensure that the join relation is computed correctly: Item 1 ensures that all matches will be included in the join result, and Item 2 that no mismatchings will be included in the result.

3.1 Security of adjustable join

In order to prove that our join scheme is secure, we need to define formally what security properties need to hold. In order for the security guarantees to be meaningful in the real-world, such a security definition needs to model the threat model (or a stronger threat model). We need to capture (a superset of) the ways in which an adversary could interact with our join scheme, and then specify what it should not be able to compute.

Modeling the real-world attacker. Recall that the threat model consists of an attacker who is passively observing the encrypted data and the join adjustments, but cannot issue queries actively. There are three aspects to take into consideration:

- The attacker may know the value corresponding to certain encryptions.
- Real data does not come from random distributions; moreover, we do not always know the distribution from which the data comes, but the adversary may know it in some cases. As such, a conservative adversary model is to let the adversary choose the data to be encrypted itself.
- The proxy does not choose the columns to be joined at random. Similarly, in a conservative attacker model, the adversary can choose which columns to be joined.

Considering these aspects, the adversary should not be able to find any new join relation between two columns from two sets of columns that have not been joined. That is, if the adversary does not know if two items from two non-joined columns are the same or not, the adversary should not be able to learn this information, despite having the knowledge described above.

Let us first present the definition intuitively. It has the form of a security game. The challenger picks a secret key by running the key generation algorithm. Consider two sets of columns S_0 and S_1, initially empty; the challenger will eventually challenge the adversary to compute a join relation between these two sets of columns, for which the challenger never provided join information to the adversary. Before the challenge, the adversary can populate these sets *adaptively* as it desires by inserting columns in S_0 or S_1, items in any of the columns, or asking for the join token of any two columns that must both be in the same set.

During the challenge, the adversary chooses two values, x_0 and x_1. The challenger then provides the encryptions of x_0 and x_1 for every column in S_0 and S_1; however, the encryptions of x_0 and x_1 for columns in S_1 are shuffled randomly, in a way unknown to the adversary (but in the same way for all columns in S_1). The adversary must determine which are the encryptions of x_0 in S_1, effectively determining a join relation between S_0 and S_1.

Security game. Consider the following security game between a challenger Ch and an adversary Adv for security parameter k:

I) The challenger runs KeyGen to obtain $(\mathsf{params}, \mathsf{sk})$ and sends params to Adv.

II) (Adaptive queries, first round:) Consider two sets of columns S_0 and S_1, initially empty, and a column index $I \in \mathcal{D}_k$, initially 0, known to both Adv and Ch. The adversary Adv adaptively queries the challenger in the following ways:

   (a) Adv sends "('add column', b_S)" to Ch for $b_S \in \{0, 1\}$;
   Ch adds I to S_{b_S} and increments I.

   (b) Adv sends "('encrypt item', m, i)" for some value $m \in \mathcal{D}_k$ and column $i \in \mathcal{D}_k$ where $i < I$;
   Ch responds with $\mathsf{Enc}(\mathsf{params}, \mathsf{sk}, m, i)$.

   (c) Adv sends "('join column', i, j)" for two columns $i, j \in \mathcal{D}_k$ both in S_0 or both in S_1;
   Ch responds with $\mathsf{Token}(\mathsf{params}, \mathsf{sk}, i, j)$.

III) Adv chooses values x_0 and x_1 with the constraint that Adv could not have asked 'encrypt item' for x_0 or x_1 for $i \in S_1$ in the previous stage, and sends them to Ch;
Ch responds with $\{\langle i, \mathsf{Enc}(\mathsf{params}, \mathsf{sk}, x_0, i), \mathsf{Enc}(\mathsf{params}, \mathsf{sk}, x_1, i)\rangle$, for all $i \in S_0\}$. Ch then flips a coin b and sends to Adv $\{\langle i, \mathsf{Enc}(\mathsf{params}, \mathsf{sk}, x_b, i), \mathsf{Enc}(\mathsf{params}, \mathsf{sk}, x_{1-b}, i)\rangle$, for all $i \in S_1\}$.

IV) (Adaptive queries, second round:) Adv can now run a second adaptive queries round as in Stage II, the differences being that (1) for every run of Stage IIa for $i \in S_1$, Ch also provides $\langle \mathsf{Enc}(\mathsf{sk}, \mathsf{params}, x_b, i)$, $\mathsf{Enc}(\mathsf{params}, \mathsf{sk}, x_{1-b}, i)\rangle$, and (2) Adv cannot ask ('encrypt item', x_0 or x_1, i) for $i \in S_1$.

V) Adv outputs a bit b' representing its guess for b.

We say that the adversary wins if Adv's queries have inputs in the correct ranges and $b = b'$.

Definition 4 (Security). *An adjustable join scheme, ADJ-JOIN, is secure if, for all polynomial-size family of circuits,* $\{\mathsf{Adv}_k\}_k$,

$$\Pr[\mathsf{Adv}_k \textit{ wins the security game of ADJ-JOIN for } k] < 1/2 + \mathsf{negl}(k),$$

where the probability is taken over KeyGen*'s coin tosses and the choice of* b*.*

4 An adjustable join construction, ADJ-JOIN

We now describe our ADJ-JOIN construction, which is relatively simple. We must specify each algorithm from Def. 2.

Let $\{\mathsf{PRP}_k\}_k$ be any family of pseudorandom permutations, with $\mathsf{PRP}_k : \mathcal{D}_k \to \mathbb{Z}_p$. Note that the existence of such pseudorandom permutations does not introduce a new assumption because one can construct pseudorandom permutations from any one-way function [2] including one based on ECDDH.

Algorithm 1 ($\mathsf{KeyGen}(1^k)$)**.**
1. Compute $\mathsf{params} = (q, E, p, P)$ by running $\mathsf{CurveGen}(1^k)$.
2. Draw two random secret keys $\mathsf{sk}_{\mathsf{col}}$ and $\mathsf{sk}_{\mathsf{msg}}$ to be used to index PRPs out of the PRP family. Let $\mathsf{sk} = (\mathsf{sk}_{\mathsf{col}}, \mathsf{sk}_{\mathsf{msg}})$.
3. Output $(\mathsf{params}, \mathsf{sk})$.

Algorithm 2 ($\mathsf{Enc}(\mathsf{params}, \mathsf{sk}, m \in \mathcal{D}_k, i \in \mathcal{D}_k)$)**.**

1. Compute $\mathsf{csk}_i = \mathsf{PRP}_{\mathsf{sk}_{\mathsf{col}}}(i)$.
2. Output $C = P \cdot \mathsf{csk}_i \cdot \mathsf{PRP}_{\mathsf{sk}_{\mathsf{msg}}}(m) \in E_P$.

Algorithm 3 ($\mathsf{Token}(\mathsf{params}, \mathsf{sk}, i \in \mathcal{D}_k, j \in \mathcal{D}_k)$)**.**

1. Compute $\mathsf{csk}_i = \mathsf{PRP}_{\mathsf{sk}_{\mathsf{col}}}(i)$ and $\mathsf{csk}_j = \mathsf{PRP}_{\mathsf{sk}_{\mathsf{col}}}(j)$.
2. Compute csk_i^{-1} to be the inverse of csk_i in $\mathbb{Z}_p$.
3. Output $t = \mathsf{csk}_i^{-1}\mathsf{csk}_j \in \mathbb{Z}_p$.

Algorithm 4 ($\mathsf{Adj}(\mathsf{params}, C, t \in \mathbb{Z}_p)$)**.**

1. Output $Ct \in E_P$.

Let us first argue that the construction is correct according to Def. 3.

Claim 1. *$\mathsf{ADJ\text{-}JOIN}$ is a correct adjustable join.*

Proof. To prove Property 1:

$$\begin{aligned}
\mathsf{Adj}(\mathsf{params}, C_0^i, t_{i\to j}) &= C_0^i t_{i\to j} \\
&= \mathsf{Enc}(\mathsf{params}, \mathsf{sk}, m_0, i)\mathsf{Token}(\mathsf{params}, \mathsf{sk}, i, j) \\
&= P \cdot \mathsf{csk}_i \mathsf{PRP}_{\mathsf{sk}_{\mathsf{msg}}}(m_0)\mathsf{csk}_i^{-1}\mathsf{csk}_j \\
&= P \cdot \mathsf{PRP}_{\mathsf{sk}_{\mathsf{msg}}}(m_0)\mathsf{csk}_j \\
&= C_0^j \ ,
\end{aligned}$$

as desired. We can also see that Property 2 holds:

$$\begin{aligned}
C_0^i &= \mathsf{Enc}(\mathsf{params}, \mathsf{sk}, m_0, i) \\
&= P \cdot \mathsf{csk}_i \mathsf{PRP}_{\mathsf{sk}_{\mathsf{msg}}}(m_0) \\
&\neq P \cdot \mathsf{csk}_i \mathsf{PRP}_{\mathsf{sk}_{\mathsf{msg}}}(m_1) \quad \text{(because } \mathsf{PRP}_{\mathsf{sk}_{\mathsf{msg}}} \text{ is a permutation)} \\
&= C_1^i \ .
\end{aligned}$$

□

4.1 Implementation

For our ADJ-JOIN implementation, we use domain space $\mathcal{D}_k = 0 \ldots 2^k - 1$. We use AES to implement PRP because AES is believed to be a pseudorandom permutation. We use a NIST curve in which ECDDH is believed to hold.

5 Security proof

Before presenting the proof for our security definition, we prove that the adversary cannot break a simpler and similar challenge; we will then naturally extend this result to the stronger adversary.

5.1 A simpler challenge

Lemma 2 (Simple challenge)**.** *If the ECDDH assumption holds for elliptic curves generated by* $\mathsf{CurveGen}$*, then, for all polynomial-size family of circuits* $\{\mathsf{Adv}_k\}_k$*,*

$$\Pr\left[\begin{array}{l}\mathsf{params} = (q, E, p, P) \leftarrow \mathsf{CurveGen}(1^k);\\ U_0, U_1 \leftarrow E_P;\\ x, y \leftarrow \mathbb{Z}_P; d_0 = x, d_1 = y;\\ b \leftarrow \{0,1\} :\\ \mathsf{Adv}_k(\mathsf{params}, U_0, U_1, U_0' = U_0x, U_0'' = U_0y, U_1' = U_1d_b, U_1'' = U_1d_{1-b}) = b\end{array}\right] < 1/2 + \mathsf{negl}(k).$$

Intuitively, we can see that the adversary's challenge in this definition is a simplification of the challenge in our security game in Def. 4 because we can think of the simpler challenge as follows. The adversary receives an encryption U_0 of an item in column 0, and an encryption U_1 of the same item in column 1. (This item can be any random number γ because U_0 and U_1 are random). Then, the adversary receives encryptions for some values v_x and v_y for column 0, U_0x and U_0y, and encryptions of the same values for column 1, U_1x and U_1y; the adversary receives the latter two encryptions in a random order given by a bit b. The adversary has to guess b, essentially deducing the join relation for v_x and v_y across columns 0 and 1.

Proof of Lemma 2. The proof proceeds by contradiction. Assuming there is a poly-size family of circuits $\{\mathsf{Adv}_k\}_k$ that guesses b nonnegligibly, we want to construct a poly-size family of circuits $\{B_k\}_k$ that breaks ECDDH. For a security parameter k, let $\mathsf{Adv} = \mathsf{Adv}_k$ and $B = B_k$.

B receives as arguments an instance of a ECDDH problem it has to break: params, P_0, P_1, $P_0' = P_0u_0$ and $P_1' = (P_1u_0$ or $P_1u_1)$ as in Def. 1. Intuitively, B should feed its inputs in some form to Adv to leverage its power to break the ECDDH problem instance.

Let $\alpha(k)$ be the probability with which Adv guesses b during the simple challenge, which is $1/2$ plus a non-negligible function. Let $\beta(k)$ be the probability that Adv guesses b on a different input distribution:

$$\beta(k) = \Pr\left[\begin{array}{l}\mathsf{params} \leftarrow (q, E, p, P) \leftarrow \mathsf{CurveGen}(1^k);\\ U_0, U_1 \leftarrow E_P;\\ x, y, z \leftarrow \mathbb{Z}_P; d_0 = x, d_1 = z;\\ b \leftarrow \{0,1\} :\\ \mathsf{Adv}_k(\mathsf{params}, U_0, U_1, U_0x, U_0y, U_1d_b, U_1d_{1-b}) = b\end{array}\right].$$

Basically, instead of providing U_1y, we provide Adv with an independent random point U_1z; the question now is whether Adv can still discover which is U_1x (the input that shares the same scalar with U_0x). Given $\{\mathsf{Adv}_k\}_k$, both $\alpha(k)$ and $\beta(k)$ have well-defined values.

We consider two cases based on the relative values of $\beta(k)$ and $\alpha(k)$; in each case, we provide a different construction of B. These two cases are comprehensive and we will see that B has nonnegligible advantage of breaking ECDDH in both of them.

Case 1: $\beta(k) < 1/4 + \alpha(k)/2$

Algorithm 5 ($B(\mathsf{params}, P_0, P_1, P_0', P_1')$)**.**

1. B draws $w \leftarrow \mathbb{Z}_p$ and $b^* \leftarrow \{0,1\}$.
2. Let $T_0 = P_1'$ and $T_1 = P_1w$.
3. B provides the following inputs to Adv:

$$(\mathsf{params}, P_0, P_1, P_0w, P_0', T_{1-b^*}, T_{b^*}).$$

4. Adv provides his guess b^{Adv} for b^*. If $b^{\mathsf{Adv}} = b^*$, output 0 meaning that B believes that P_1' equals $P_1 u_0$, else output 1 meaning that P_1' is random.

Let's argue that B has nonnegligible probability of breaking ECDDH:

$$\begin{aligned}
\Pr[B \text{ wins}] &= 1/2(\Pr[B \text{ wins}|P_1' = P_1 u_0] + \Pr[B \text{ wins}|P_1' = P_1 u_1]) \\
&= 1/2\,(\alpha(k) + 1 - \beta(k)) \\
&> 1/2 + 1/4(\alpha(k) - 1/2), \quad \text{(by the condition on } \beta(k) \text{ from Case 1)}
\end{aligned}$$

which is nonnegligibly larger than $1/2$.

Case 2: $\beta(k) \geq 1/4 + \alpha(k)/2$

Algorithm 6 ($B(\mathsf{params}, P_0, P_1, P_0', P_1')$)**.**

1. B draws $w, v \leftarrow \mathbb{Z}_p$ and $b^* \leftarrow \{0,1\}$.
2. Let $T_0 = P_1'$ and $T_1 = P_1 v$.
3. B provides the following inputs to Adv:

$$\left(\mathsf{params}, P_0, P_1, P_0', P_0 w, T_{b^*}, T_{1-b^*}\right).$$

4. Adv provides his guess b^{Adv} for b^*. If $b^{\mathsf{Adv}} = b^*$, output 0, else output 1.

Let's argue that B has a nonnegligible probability of breaking ECDDH. First note that $\Pr[B \text{ wins}|P_1' = P_1 u_1] = 1/2$ because B has information theoretically no knowledge about b' because it receives two random numbers $P_1 v$ and $P_1 u_1$ independent of the other values provided to B.

$$\begin{aligned}
\Pr[B \text{ wins}] &= 1/2(\Pr[B \text{ wins}|P_1' = P_1 u_0] + \Pr[B \text{ wins}|P_1' = P_1 u_1]) \\
&= 1/2\,(\beta(k) + 1/2) \\
&\geq 1/2 + 1/4(\alpha(k) - 1/2), \quad \text{(by the condition on } \beta(k) \text{ from Case 2)}
\end{aligned}$$

which is nonnegligibly larger than $1/2$.

In conclusion, B has nonnegligible advantage at breaking ECDDH, which concludes our proof.

□

5.2 Random oracle weak-security

In this section, we move closer to our proof of security. We prove a weaker security notion, denoted *weakly-secure adjustable join* for a *random oracle* $\mathsf{ADJ\text{-}JOIN}$. The random oracle $\mathsf{ADJ\text{-}JOIN}$ is our $\mathsf{ADJ\text{-}JOIN}$ scheme where $\mathsf{PRP}_{\mathsf{sk}_{\mathsf{msg}}}$ and $\mathsf{PRP}_{\mathsf{sk}_{\mathsf{col}}}$ are replaced with two independent random oracles, $\mathsf{O}^k_{\mathsf{msg}}$ and $\mathsf{O}^k_{\mathsf{col}}$, respectively.

Definition 5 (Weak-security). *A weakly-secure adjustable join is defined the same as in Def. 4, except that* Adv *is not allowed to ask for encryptions of* x_0 *and* x_1 *in Step IIb of the first round of adaptive queries even for columns in* S_0.

Theorem 3. *If the ECDDH assumption holds, random oracle* $\mathsf{ADJ\text{-}JOIN}$ *is a weakly-secure adjustable join.*

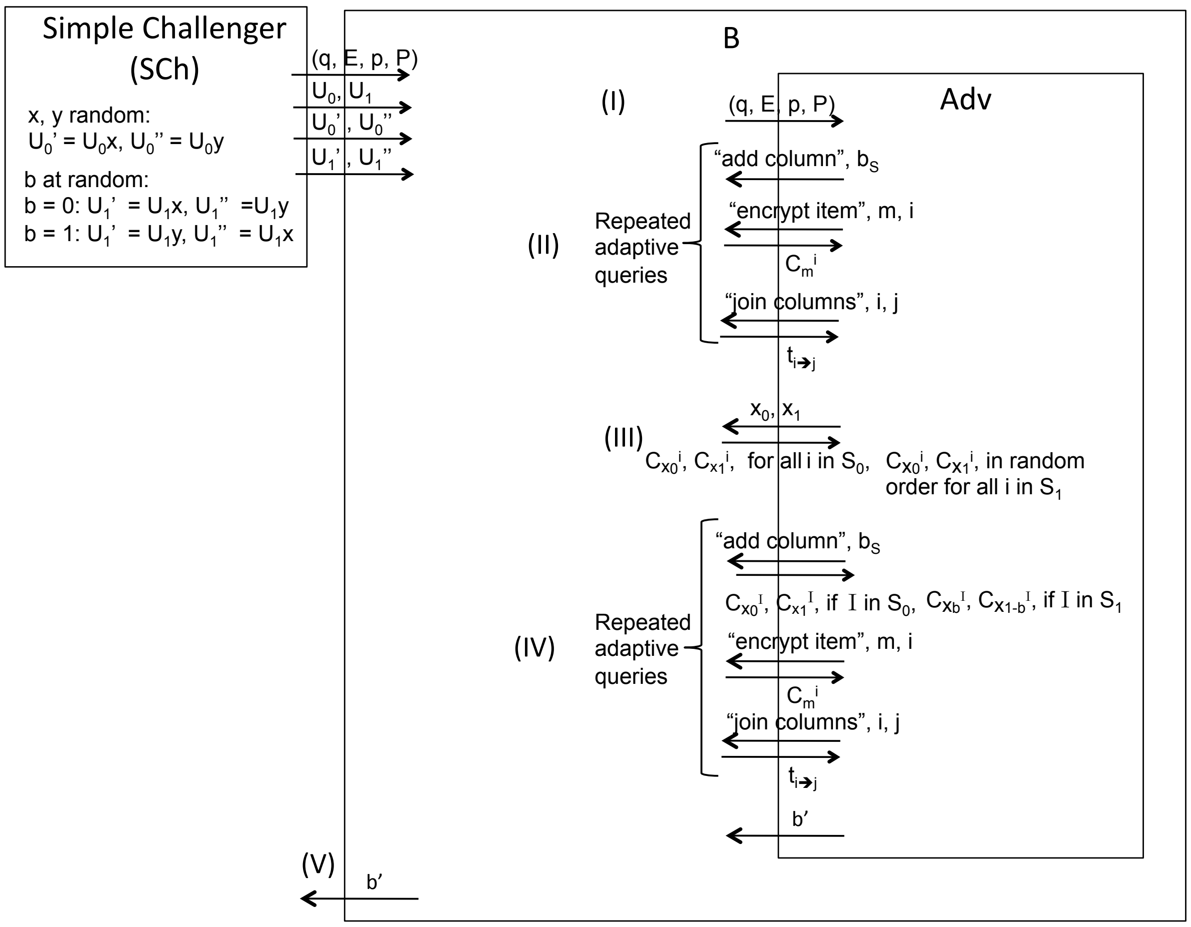

Figure 2: Sketch of the setup and notation for the reduction B.

Proof. We proceed by contradiction. Assume there is a poly-size family of circuits $\{\mathsf{Adv}_k\}_k$ having nonnegligible probability of winning when interacting with the challenger Ch from Def. 5's security game; let us construct a polynomial-size family of circuits $\{B_k\}_k$ that can solve the simple challenge (Lemma 2) when interacting with challenger SCh. By Lemma 2, it follows that one can break the ECDDH assumption.

For a given k, let $\mathsf{Adv} = \mathsf{Adv}_k$ and $B = B_k$. Figure 2 shows a sketch of the notation we will use by showing the inputs corresponding to an instance of the simple challenge that B receives, the communication that B must have with Adv, and the bit b' that B must guess. As we describe B, the figure will also help the reader keep track of the communication between B and Adv.

B must transform its inputs to feed inputs to Adv so that it can leverage Adv's power to break the simple challenge. Here is how B works.

Algorithm 7 (Reduction $B(\mathsf{params}, P_0, P_1, P_0', P_1')$)**.**

In the following stages, we will only treat cases in which Adv provides B values and queries that are within the allowed ranges as defined in the security game; if Adv does otherwise, B simply outputs a random guess: $b' \leftarrow \{0, 1\}$.

Stage (I)

1. B simply forwards params to Adv.

Stage (II)

2. Assume that the first column Adv adds to S_0 is column 0 and the first column it adds to S_1 is 1; the

reduction can straightforwardly be modified otherwise. B maintains I the same way as Ch does, so I is now 2.

3. B chooses csk_0 and csk_1 keys as follows:

$$P \cdot \mathsf{csk}_0 := U_0, \text{ and } P \cdot \mathsf{csk}_1 := U_1. \tag{3}$$

This means that, for example, whenever B wants to use $P\mathsf{csk}_0$, it will use U_0. B cannot compute csk_0 and csk_1 because computing the discrete log is hard, but csk_0 and csk_1 are well-defined values; the insight here is that B will only need $P\mathsf{csk}_0$ and not necessarily csk_0. Let's see how B answers each of Adv's challenges:

- Adv *requests ("add column", b_S) for column I:* B generates $d_{b_S \to I} \leftarrow \mathbb{Z}_p$ and sets:

$$P \cdot \mathsf{csk}_I := P \cdot \mathsf{csk}_{b_S} \cdot d_{b_S \to I} = U_{b_S} d_{b_S \to I} \tag{4}$$

  For example, if $b_S = 0$ (meaning I should be added to S_0), B will use $U_0 d_{0 \to I}$ whenever it needs to use $P\mathsf{csk}_I$; that is, instead of generating a random csk_I (as in Alg. 2, Step 1), B generates a random "delta" $d_{0 \to I}$ with respect to csk_0 and computes $d_{0 \to I}\mathsf{csk}_0$, which is equivalent. For completeness, let $d_{0 \to 0} = 1$ and $d_{1 \to 1} = 1$.

- Adv *requests ("encrypt", m, i):* Let b_S be the bit indicating if i is in S_0 or S_1. B returns $U_{b_S} d_{b_S \to i} \cdot \mathsf{O}^k_{\mathsf{msg}}(m)$.

- Adv *requests ("join", i, j), where i and j are both in S_0 or both in S_1:* without loss of generality, let us consider that $i, j \in S_0$. B returns $d^{-1}_{0 \to i} \cdot d_{0 \to j} \in \mathbb{Z}_p$.

Stage (III)

4. B receives x_0 and x_1 from Adv.
5. Let $i \in S_0$. B provides $C^0_{x_0} = U'_0$ and $C^0_{x_1} = U''_0$ to Adv. B could have naturally computed $C^0_{x_0}$ to be $P\mathsf{csk}_0\mathsf{O}^k_{\mathsf{msg}}(x_0)$ as the Enc procedure does; instead, B uses part of his challenge inputs. The intuition here is that $\mathsf{O}^k_{\mathsf{msg}}(x_0)$ returns a random value independent of x_0 and U'_0 was also obtained by multiplying a random value to U_0. To generalize for $i \in S_0$, B provides $C^i_{x_0} = U'_0 d_{0 \to i}$ and $C^i_{x_1} = U''_0 d_{0 \to i}$. For $i \in S_1$, B similarly computes C^i_*: it sends $\langle U'_1 d_{1 \to i}, U''_1 d_{1 \to i} \rangle$ in this order.

Stage (IV)

6. B answers queries from Adv in the same way as at Stage (II), except that it computes encryptions for x_0 and x_1 for $i \in S_0$ and provides values for $C^I_{x_*}$ for new columns in S_1 using the method in Stage (III).

Stage (V): Decision

7. B receives Adv's guess b' and it simply forwards this bit to SCh.

Now that we presented B, we need to argue that $\{B_k\}_k$ is a poly-size family of circuits and that B breaks the simple challenge nonnegligibly. Since $\{\mathsf{Adv}_k\}_k$ is a poly-size family of circuits and $\{B_k\}_k$ only does polynomial additional amount of work, the first property follows.

Let us examine the chance that B will guess b' correctly. First note that if Adv guesses correctly, then B guesses correctly as well.

Now let's argue that the distribution of inputs Adv receives is indistinguishable from the one it receives when interacting with Ch in the random oracle model; in this case, Adv has nonnegligible advantage of winning. We proceed stage by stage.

Stage (I): SCh uses the same $\mathsf{CurveGen}$ algorithm as Ch, so the distributions are equal.

Stage (II): Ch picks csk_i at random using a random oracle. Since SCh chooses U_0 and U_1 at random from E_P, based on Eq. 3, the induced csk_0 and csk_1 will also be uniformly distributed over $\mathbb{Z}_p$. Now since all $d_{b_S \to I}$ values are chosen at random from $\mathbb{Z}_p$, similarly all csk_I values will be random in $\mathbb{Z}_p$ so indistinguishable from outputs of a random oracle. Because of this, all encryptions returned by B are indistinguishable from encryptions from Ch.

Regarding the "join" response, we can see that $d_{0\to i}^{-1} \cdot d_{0\to j} = \mathsf{csk}_i^{-1}\mathsf{csk}_j$ for $i, j \in S_0$ (and similarly for S_1) as desired. Note that B would not have been able to compute this step correctly if Adv were allowed to ask for a join between S_0 and S_1, because a value such as $d_{0\to i}d_{1\to j}$ would have been incorrect.

Stage (III): Consider how B computes $C_{x_0}^i$ for $i \in S_0$. If B were to provide $P\mathsf{csk}_i\mathsf{O}_{\mathsf{msg}}^k(x_0)$ to Adv, this value would certainly have the right distribution because this is the same value Ch would compute. Instead, B provides $C_{x_0}^0 = U_0'$. Note that since SCh chose x at random, this makes it indistinguishable from $\mathsf{O}_{\mathsf{msg}}^k(x_0)$. The key point here is that Adv has not seen any encryptions of x_0 before due to the definition of weakly-secure adjustable join. The same argument applies to $C_{x_0}^i$ and $C_{x_1}^i$ for all $i \in S_0$.

Since $P\mathsf{csk}_1$ equals U_1, the same argument applies to $C_{x_0}^i$ and $C_{x_1}^i$ for all $i \in S_1$. Here, the ciphertexts for x_0 and x_1 are already ordered randomly by the random bit used by SCh.

Stage (IV): By the same argument we presented in Stage (II) and Stage (III), we can see that Adv will receive inputs with the right distribution.

Therefore, overall, B receives inputs for which it can break random oracle $\mathsf{ADJ\text{-}JOIN}$ with nonnegligible probability which means that Adv will also break the simple challenge with nonnegligible probability, which reaches a contradiction based on Lemma 2. □

5.3 Plain model security

Let us now go back and prove the actual (stronger) security definition, Def. 4.

Theorem 4. *If the ECDDH assumption holds and if each* PRP *invoked in* $\mathsf{ADJ\text{-}JOIN}$ *is a random oracle,* $\mathsf{ADJ\text{-}JOIN}$ *is a secure adjustable join.*

Proof. The proof of this theorem involves making one modification to the proof of weak-security (Theorem 3). The same proof would not work because of one issue: in order to argue that, in Stage (III), Adv receives the right input distribution when receiving encryptions of x_0 and x_1, we crucially relied on the fact that Adv has not requested encryptions for x_0 or x_1 for a column in S_0 in Stage (II). The reason is that encryptions B provides to Adv in Stage (II) may not match those in Stage (III), which Adv can easily detect. But the original security model allows B to ask for such encryptions.

To ensure that Adv still gets the desired distribution over the inputs, B needs to insert $C_{x_0}^0 := U_0'$ in Stage (II) the first time that Adv asks for an encryption of x_0. But how can B know that a certain m it receives for encryption in Stage (II) will be equal to x_0 from Stage (III)?

The idea is for B to guess randomly which could these values be; this is possible with nonnegligible probability because Adv makes at most a polynomial number of such queries.

Since $\{\mathsf{Adv}_k\}_k$ is a poly-size family of circuits, there exists a polynomial $\alpha(k)$ in the security parameter k that is always larger than the number of queries Adv makes in Stage (II). Consider four cases: Adv asks for x_0 and x_1 in Stage (II), Adv asks only for x_0 in Stage (II), Adv asks only for x_1 in Stage (II), and Adv asks for neither x_0 or x_1 in Stage (II). B uses the following prediction algorithm:

- Guess which case will happen, by considering that each case occurs with probability $1/4$.

- If Adv does not ask for either x_0 or x_1 in Stage (II) in the case predicted, use the same reduction as in the proof of Theorem 3. Otherwise, B considers the maximum of queries that Adv may make, $\alpha(k)$. It then considers a uniform distribution over $\alpha(k)$ queries and samples from the distribution to guess in which queries will Adv request x_0 and/or x_1. If B predicts that a certain query asks for encryption x_0, B will computes the encryption using the procedure in Stage (III).

When B reaches Stage (III) in its conversation with Adv and learns x_0 and x_1 from Adv, B will be able to check if its prediction was correct. If it was not, B guesses b at random, else forwards Adv's guess as before.

Whenever B predicts correctly, B has a nonnegligible probability of guessing b. Also B predicts correctly with a probability on the order of $1/\alpha^2(k)$. Therefore, B still has overall nonnegligible probability of breaking the simple challenger, as desired.

□

Our final result now follows easily:

Corollary 5 (Final result). *If the ECDDH assumption holds, ADJ-JOIN is a secure adjustable join scheme.*

Proof. The secret keys $\mathsf{sk}_{\mathsf{col}}$ and $\mathsf{sk}_{\mathsf{msg}}$ are chosen at random and never released to the adversary. All the adversary may ever get to see is a mapping between certain input values and output values of these PRPs (in fact, the adversary never receives this information in the clear, but for worst-scenario's sake, we can consider this). This means that all the adversary has is an oracle to each of these PRPs. By the definition of a PRP (see [2]), no PPT adversary can distinguish non-negligibly between a random oracle and a PRP oracle, so the adversary gains only negligible probability at winning the security game on top of the negligible probability from Theorem 4. □

6 Adjustable pseudorandom permutation

Our ADJ-JOIN construction could be used outside of CryptDB because of the simple functionality it provides. Intuitively, it is a pseudorandom permutation whose key can be adjusted to a different key by an external party not knowing the secret key. We prove some general useful properties about our construction that are likely to be applicable to other settings:

- Our construction induces a PRP family.
- An external party can adjust the key between two PRPs.
- Despite such adjustability, two PRPs are independent. This property is not straightforward because, since two PRPs could be adjusted to each other, they may have information in common (as is the case with part of the secret key in our construction). Such information may cause a correlation and may enable an adversary to distinguish a pair of such functions from a pair of independent random oracles, even if the adversary could not distinguish any such function individually from a random oracle.
- Even after adjusting one PRP to another, the second scheme remains a secure PRP.

The rest of this section formalizes these aspects.

6.1 Construction: ADJ-PRP

Our adjustable PRP scheme is called ADJ-PRP. Despite the great similarity to ADJ-JOIN, we give this scheme a different name and present its construction because PRPs have a different form than adjustable join schemes (2).

Algorithm 8 ($\mathsf{KeyGen}^*(1^k)$)**.**

1. Compute $\mathsf{params} = (q, E, p, P)$ by running $\mathsf{CurveGen}(1^k)$.
2. Draw a secret key sk_m to be used to index PRPs out of the PRP family.
3. Output $(\mathsf{params}, \mathsf{sk}_m)$.

Algorithm 9 ($\mathsf{AdjPRP}_{\mathsf{params},\mathsf{sk}_m,\mathsf{sk}_c}(m \in \mathcal{D}_k)$)**.**

1. Output $P \cdot \mathsf{sk}_c \cdot \mathsf{PRP}_{\mathsf{sk}_m}(m) \in E_P$.

Algorithm 10 ($\mathsf{Token}^*(\mathsf{params}, \mathsf{sk}_0, \mathsf{sk}_1)$)**.**

1. Compute sk_0^{-1} to be the inverse of sk_0 in $\mathbb{Z}_p$.
2. Output $t = \mathsf{sk}_0^{-1}\mathsf{sk}_1 \in \mathbb{Z}_p$.

Adj is the same for AdjPRP as for ADJ-JOIN (Alg. 4).

6.2 Proving security

Claim 6 (AdjPRP is a PRP)**.** *If* PRP *is a PRP, the function ensemble* $\{(\mathsf{params}, \mathsf{sk}_m) \leftarrow \mathsf{KeyGen}^*(k) : \mathsf{sk}_c \leftarrow \mathbb{Z}_p : \mathit{AdjPRP}_{\mathsf{params},\mathsf{sk}_m,\mathsf{sk}_0}\}_k$ *is a pseudorandom permutation ensemble.*

Proof. Multiplying elements of E_P by $P \cdot \mathsf{sk}_c$ induces a permutation. Therefore, we apply a permutation to PRP which is a pseudorandom permutation; the result is thus a PRP as well, and the claim follows. □

Claim 7 (Correctness of adjustability.)**.** *For all* k*, for all* $(\mathsf{params}, \mathsf{sk}_m)$ *produced by* $\mathsf{KeyGen}^*(1^k)$*, for all* $\mathsf{sk}_0, \mathsf{sk}_1 \in \mathbb{Z}_p$*, for all* m_0*,* $m_1 \in \mathcal{D}_k$ *with* $m_0 \neq m_1$*,* $C_b^l = \mathit{AdjPRP}_{\mathsf{params},\mathsf{sk}_m,\mathsf{sk}_l}(m_b)$ *for* $l, b \in \{0, 1\}$ *and* $t = \mathsf{Token}^*(\mathsf{params}, \mathsf{sk}_0, \mathsf{sk}_1)$*, it holds that* $\mathsf{Adj}(\mathsf{params}, C_0^i, t) = C_0^j$*, and* $C_0^i \neq C_1^i$*.*

Proof. The proof is the same as for Claim 1. □

The definition of a PRP requires that the secret key of a PRP be chosen freshly at random to guarantee indistinguishability from a random oracle. In our case though, in order to use adjustability, only the sk_c part of the key can be chosen freshly at random, while sk_m remains common between two instances. Therefore, we need to prove that the two functions will not be correlated: that is, no adversary can distinguish between a pair of such functions and a pair of independent random oracles.

Theorem 8 (Random oracle pair indistinguishability.)**.** *Under the ECDDH assumption, for all poly-size circuits* Adv*, for all sufficiently large* k*,*

$$\Pr\left[\begin{array}{l} (\mathsf{params}, \mathsf{sk}_m) \leftarrow \mathsf{KeyGen}^*(1^k), \mathsf{sk}_0, \mathsf{sk}_1 \leftarrow \mathbb{Z}_p; \\ \mathcal{O}_0 = (\textit{oracle for } \mathit{AdjPRP}_{\mathsf{params},\mathsf{sk}_m,\mathsf{sk}_0}, \textit{oracle for } \mathit{AdjPRP}_{\mathsf{params},\mathsf{sk}_m,\mathsf{sk}_1}); \\ \mathcal{O}_1 = (\textit{random oracle, random oracle}); \\ \mathsf{Adv}(\mathsf{params}, \mathcal{O}_b, \mathcal{O}_{1-b}) = b \end{array}\right] < \frac{1}{2} + \mathsf{negl}(k).$$

Proof. Similarly to the proof of Corollary 5, we can assume that each PRP with a different random key in the construction of AdjPRP can be replaced with a random oracle because of the security properties of PRPs. Therefore, the oracle for $\mathsf{AdjPRP}_{\mathsf{params},\mathsf{sk}_m,\mathsf{sk}_0}$ is indistinguishable from the first random oracle. For the same input x, the answers of the $\mathsf{AdjPRP}_{\mathsf{params},\mathsf{sk}_m,\mathsf{sk}_0}$ will be a factor of ($\mathsf{sk}_0 \cdot \mathsf{sk}_1^{-1} \in \mathbb{Z}_p$) of the answers of the oracle $\mathsf{AdjPRP}_{\mathsf{params},\mathsf{sk}_m,\mathsf{sk}_1}$.

Since $t = \mathsf{sk}_0 \cdot \mathsf{sk}_1^{-1} \in \mathbb{Z}_p$ is uniformly distributed in $\mathbb{Z}_p$, the adversary basically has to distinguish between a pair of sets with random points $(P_0, P_1, \dots, ; R_0, R_1, \dots)$ and a pair consisting of a set with random points and the same set multiplied by a random scalar $(P_0, P_1, \dots ; P_0 t, P_1 t, \dots)$. This is the extended form of ECDDH because it has the same form as ECDDH but instead it has a polynomial number of terms; the extended ECDDH has been shown reducible to the basic ECDDH, thus ensuring that the Adv cannot guess nonnegligibly.

$\square$

We now prove that, even though the adversary receives the adjustment token between two instances of AdjPRP, each instance by itself remains a PRP.

Theorem 9 ($\mathsf{ADJ\text{-}PRP}$ remains a PRP after adjustment)**.** *If* PRP *is a PRP, for all poly-size circuits* Adv*, for all k sufficiently large,*

$$\Pr\left[\begin{array}{l} (\mathsf{params}, \mathsf{sk}_m) \leftarrow \mathsf{KeyGen}^*(1^k); \mathsf{sk}_0, \mathsf{sk}_1 \leftarrow \mathbb{Z}_p; \\ t = \mathsf{Token}^*(\mathsf{params}, \mathsf{sk}_0, \mathsf{sk}_1); \\ \mathcal{O}_0 = (\textit{oracle for } \mathsf{AdjPRP}_{\mathsf{params},\mathsf{sk}_m,\mathsf{sk}_1}); \\ \mathcal{O}_1 = (\textit{random oracle}); \\ \mathsf{Adv}(t, \mathcal{O}_b, \mathcal{O}_{1-b}) = b \end{array}\right] < \frac{1}{2} + \mathsf{negl}(k).$$

Proof. Since sk_1 is a uniformly random value in $\mathbb{Z}_p$, $t = \mathsf{sk}_0^{-1}\mathsf{sk}_1 \in \mathbb{Z}_p$ is also a uniformly random value and it is information theoretically independent of sk_0. Since Adv does not receive sk_1, t does not provide any useful information to Adv. This means that Adv receives the same information from its inputs as in the case of Claim 6. Therefore, this means that there exists Adv' such that

$$\text{Probability } \mathsf{Adv} \text{ guesses right} \le \Pr\left[\begin{array}{l} (\mathsf{params}, \mathsf{sk}_m) \leftarrow \mathsf{KeyGen}^*(1^k); \mathsf{sk}_0 \leftarrow \mathbb{Z}_p; \\ \mathcal{O}_0 = (\text{oracle for } \mathsf{AdjPRP}_{\mathsf{params},\mathsf{sk}_m,\mathsf{sk}_1}); \\ \mathcal{O}_1 = (\text{random oracle}); \\ \mathsf{Adv}'(\mathcal{O}_b, \mathcal{O}_{1-b}) = b \end{array}\right],$$

because the token t does not provide any useful information. This latter probability is at most $1/2 + \mathsf{negl}(k)$ because AdjPRP is a PRP by Claim 6. $\square$

References

[1] Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *Lecture Notes in Computer Science*, pages 13–25. Springer-Verlag, 1998.

[2] Oded Goldreich. *Foundations of Cryptography*, volume Basic Tools. Cambridge University Press, 2001.

[3] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987.

[4] Tatsuaki Okamoto, Eiichiro Fujisaki, and Hiraku Morita. PSEC: Provably secure elliptic curve encryption scheme (Submission to p1363a). *IEEE P1363a*, 1999.

[5] Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. CryptDB: Protecting confidentiality with encrypted query processing. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles*, Cascais, Portugal, October 2011. 14 pages.