MSc Thesis

Proofs of secure Erasure

Nikolaos P. Karvelas

supervised by:

Aggelos Kiayias

To Agne, Panayiotis, Maria and G.A.

Acknowledgements

This master Thesis has been made possible with the help of many people. First and most importantly I would like to thank my supervisor Prof. Dr. Aggelos Kiayias for introducing me to a field of science, namely Cryptology, that uses abstract Mathematics in order to provide society with the necessary tools and services to realize human ideals in our new technological era. In the many hours that Prof. Dr. Kiayias has dedicated in communicating to me (and at his classes to the rest of his students) the new paths that the cryptographic community is treading now, he has helped me see that Mathematics and theoretical Computer Science are not mere mind constructions that exist in an ideal world, but are actually necessary tools, that can be used for the benefit of all in a practical level.

Furthermore I would like to thank the other members of the committee, Prof. Dr. Aris Pagourtzis, Prof. Dr. Stathis Zachos and Prof. Dr. Vassilios Zisimopoulos for their support and helpful comments.

I would also like to thank all the members and Professors of the Master Programm in Logic, Algorithms and Computation Theory for giving me the honour of participating in the Program and learning under their wise guidance.

Moreover I would like to thank my wife Agne for all her support in the past years and for providing me wholeheartedly with the necessary psychological support and serenity that are necessary for one in order to begin a journey in the higher realms of Science. In the same manner I would like to thank my parents, Panayiotis and Maria, for supporting all my decisions and quests in all my years of Study, whether it was Music or Mathematics.

Last but not least I would like to thank my dear friends Dr. Kyriakos Kampoukos and Prof. Dr. Joseph Papadatos for being wonderful role-models and for opening my eyes to the world of Ideals.

Abstract

We investigate the problem of verifying the internal state of a remote embedded device (remote attestation), using what was by Perito and Tsudik introduced as Proofs of Secure Erasure. This is a procedure that has to take place in many cases, ranging from wireless sensor networks to any device running a software update: One has to make sure that even a compromised device will erase all of its memory contents when asked to, leaving no part of it left unaltered, possibly running malicious software. The protocols proposed thus far demand either very high communication complexity or very high time complexity that renders them ineffective for most practical applications.

Contents

Chapter 1

Introduction

Attacks on individual devices can be carried out either physically or remotely. In either case it is therefore necessary that a trusted party (called here *verifier*) be able to determine, whether the device has been compromised or not. The scenarios in which such a demand arises are many. Consider for example the process of code update: The device updated, must first erase all of its contents before performing the update. How can one be sure though that the machine has really deleted all of its contents and has not left a potential piece of malware, that can still remain and/or replicate itself in the new installed software? As a second (and potentially more critical) example consider the case of wireless sensor or actuator network, where the verifier wants to examine which (if any at all) nodes of the network have been compromised. The suspected nodes would be asked by the verifier to erase all their memory contents and then perform an update. Knowing however that there exists malware as small as 13Kb and that networks like the ones mentioned above can have critical role (utility distribution networks, industrial control systems etc.), urge for the creation of cryptographic primitives, that will be able to guarantee security for this problem.

A first attempt in solving this problem was made by Perito and Tsudik in [PT10]. Their solution is a protocol, that they name Proof of Secure Erasure (PoSE) and works as follows: The verifier sends to the prover random data, as large as the latter's memory capacity. The prover calculates a keyed Hash function (known to both parties), using as key the last bits of the received data and sends it back to the verifier. The verifier calculates the same function, on the same data and with the same key and therefore a comparison of the two results convinces him that the prover has erased all its memory. [1] This solution of course solves the problem while at the same time achieving a very low computational complexity (just finding a keyed hash). The prover's memory will be exhausted in holding the data

[1] the authors take of course into account the fact that the hash functions based on the Merkle-Damgård construction should not be used.

sent to him. It is clear however that the protocol's communication complexity is so high, that renders it impractical.

Later in [DKW11] the authors suggest a different better solution, that minimizes the communication complexity. The idea behind their approach is that the verifier will send a small "seed" to the prover, which the latter will "unravel" into a construction that in order to hold, the prover will have to use all its available storage. More specifically the prover has the description of a hash function and receives from the verifier an initial value. He will calculate the hash function recursively in a way described by a diamond like directed acyclic graph: Every node has constant in- and out-degrees except for the node at the top and the one at the bottom of the pyramid. The one at the top is the output value of the whole calculation and has out-degree 0 and the one in the bottom is given by the verifier and has in-degree 0. This solution reduces the communication complexity to the minimum, but demands a quadratic computational complexity on the size of the graph. That said, it is clear that even for a device having 1GB of memory the protocol would need so much time to run, that would render it impractical.

In our search for a better solution we came up with two protocols, that both succeed in attaining a computational complexity of $O(n \log n)$ (where n is the prover's storage capacity). The first one (named invert-Hash PoSE or iHash) is described in detail (3.3) and the second (named graphPoSE) in (3.4). The idea behind the iHash PoSE is that the prover wants to invert a hash function on a given point sent to him by the verifier. The best way he can do that is through a "clever" exhaustive search of the hash function's table. This search is done using the algorithm devised by Horowitz in [HS74], which we describe and analyze in (2.1.2), while the security proof is based on the results by Trevisan et. al. in [Rab10]. What however guided us in finding a second protocol to solve the problem, was the fact that the adversarial model for the iHash PoSE, for which we prove it secure, is quite weak, in the sense that we take into account only adversaries that work in the following way: The adversary acts in two distinct phases: A preparation phase, where he can perform any number of queries to the oracle and the actual "attack" phase, where he can no longer perform any queries to the oracle but he can use the results from the preparation phase as an advice, in order to invert the hash function on the given point.

Our second approach treads on a similar path like the one in [DKW11]: We want to find a directed acyclic graph with constant in- and out-degree, on every node of which we will calculate a function. We want also that the prover can calculate the function on this graph within a reasonable timeframe (where reasonable here would be anything less that quadratic), using all his available memory, while any adversary who would use less than all his memory, would either not be able at all to calculate the function or at least would have to pay in time that would be in at least quadratic.

Chapter 2

Preliminaries

2.1 Algorithms

In this section we present the basic algorithm that we will be using in the construction of our Invert-Hash-PoSE in 3.3. This algorithm is owed to Horowitz and Sahni in [HS74]. Their idea was later in [SS81] investigated further and improved upon. To our needs Horowitz & Sahni's algorithm is fitted perfectly; however due to the generic nature and the fact that even now Shamir & Schroeppel's algorithm still remains optimal in solving the problem at hand optimally, we find it useful to present here the basic ideas of their paper as well.

We begin with some definitions, that will help us put the problem and its solution to a wider perspective.

Definition 2.1.0.1. A problem of size n is a predicate P over $n-$bit binary strings. A string x is a solution (or a witness) of the problem, if $P(x)$ is true. The goal is to find one such x, if it exists.

Definition 2.1.0.2. A binary operator $\oplus$ on problems is a composition operator, if:

1. It is *additive*: for all P' and P'', $|P' \oplus P''| = |P'| + |P''|$[1]
2. It is *sound*: for any two solutions x' of P' and x'' of P'', the string concatenation $x'x''$ is a solution of $P' \oplus P''$
3. It is *complete*: for any solution x of P and for any representation of x as $x = x'x''$, there are problems P' and P'' such that x' solves P', x'' solves P'' and $P = P' \oplus P''$
4. It is *polynomial*: the problem $P' \oplus P''$ can be calculated in time which is polynomial in the sizes of P' and P''

[1] where $|P|$ is the problem size, defined as the number of bits in its solution

Definition 2.1.0.3. A pair of problems P' and P'' is said to be a *decomposition* of P if $P' \oplus P'' = P$.

Definition 2.1.0.4. A set of problems is polynomially enumerable if there is a polynomial time algorithm which finds for each bit string x the subset of problems which are solved by x.

Definition 2.1.0.5. A composition operator $\oplus$ is *monotonic* if the problems of each size can be totally ordered in such a way that $\oplus$ behaves monotonically, i.e. $|P'| = |P''|$ and $P' < P''$ imply that $P' \oplus P < P'' \oplus P$ and $P \oplus P' < P \oplus P''$

Definition 2.1.0.6. Given k problem/ solution tables T_i with $O(2^{n/k})$ solvable problems each, a monotonic composition operator $\oplus$, and a problem P, the *$k-$table problem* is to determine whether there are k representatives $P_i \in T_i$ such that

$$P = P_1 \oplus P_2 \oplus \cdots \oplus P_k$$

under a given parenthesization

We will now apply the above definitions, to an NP-hard problem that we will be dealing with and is very well known and studied in the literature, namely the SUBSET-SUM$((b_i)_{i=1}^n, B)$ problem, which is defined as follows:

Definition 2.1.0.7. Given an input of n positive integers $b_1, b_2, \ldots, b_n$ and a goal sum B, decide whether there exists some subset of the b_i that add up to B.

Definition 2.1.0.8. Given an input of a $k \times m$ table and a goal sum S the k-Table-SUM(S) problem decides if k integers can be chosen from this table, exactly one from each row, that add up to S.

2.1.1 Horowitz & Sahni

Horowitz and Sahni in [HS74] solve the SUBSET-SUM$((b_i)_{i=1}^n, B)$ problem, by splitting the instance into two parts, the first containing $b_1, b_2, \ldots, b_{\lfloor n/2 \rfloor}$ and the second containing $b_{\lfloor n/2+1 \rfloor}, \ldots, b_n$ and then tabulating all the target values that can be generated by summing a subset of each part into two different tables T_1 and T_2.

Each table can be computed in $O^*(2^{n/2})$ [2] time using $O^*(2^{n/2})$ space and the SUBSET-SUM$((b_i)_{i=1}^n, B)$ instance has a YES answer if and only if the constructed 2-Table-SUM(S) instance with $S = B$ has a YES answer. Based on this idea, they prove that SUBSET-SUM$((b_i)_{i=1}^n, B)$ can be solved in $O^*(2^{n/2})$ time and in $O^*(2^{n/2})$ space. Their algorithm follows:

[2] For a positive real constant c, we write $O^*(c^n)$ for a computational complexity of the form $O(c^n \cdot \text{poly}(n))$

Algorithm 1 Horowitz and Sahni

```
1: Let P be the target value
2: Sort T_1 into increasing problem order{increasing problem order has to do with the fact,
   that we have problem solutions in the tables}
3: Sort T_2 into decreasing problem order
4: while T_1 ≠ ∅ ∨ T_2 ≠ ∅ do
5:    S ← T_1.first ⊕ T_2.first {⊕ is used since in every table we have a solution to a smaller
      problem, so practically we are adding problems}
6:    if S = P then
7:       return 1 {Indicating success on solving the problem}
8:    end if
9:    if S < P then
10:      delete T_1.first from T_1
11:   end if
12:   if S > P then
13:      delete T_2.first from T_2
14:   end if
15: end while
16: return 0 {Indicating that the problem has no solution}
```

Proof. In order to prove the correctness of the algorithm, it suffices to show that whenever a problem is deleted from T_1 or T_2, it cannot possibly participate in any sum which equals P. Since T_2 is decreasing and $\oplus$ is monotonic, we have that

$$T_1.\text{first} \oplus P_2 \leq T_1.\text{first} \oplus T_2.\text{first}$$

for any $P_2 \in T_2$ and therefore the left-hand side cannot be equal to P if the right-hand side is smaller than P, justifying the deletion of T_1.first from T_1. Similarly the deletion of T_2.first from T_2 is justified. The time complexity of the sorting step is $O(2^{n/2}n/2) = O^*(2^{n/2})$ and the time complexity of the search step is $O(|T_1| + |T_2|) = O(2^{n/2})$ since at each iteration at least one element is deleted. The space complexity is $O(2^{n/2})$ since we need $2 \cdot 2^{n/2}$ space to store the 2 tables T_1 and T_2. ∎

The former result can be extended in the 3- and 4- table cases:

Theorem 2.1.1.1. *The 3-table problem can be solved in $O(2^{2n/3})$ time and $O(2^{n/3})$ space.*

Proof. For each of the $O(2^{n/3})$ problems $P_i \in T_1$, one can use the algorithm described above on the tables T_2 and T_3 in order to find a solution for $P = P_1 \oplus (P_2 \oplus P_3)$ in time $O(2^{2n/3})$ and space $O(2^{n/3})$. ∎

Theorem 2.1.1.2. *The 4-table problem with a nonbalanced parenthesis structure* $P = P_1 \oplus (P_2 \oplus (P_3 \oplus P_4))$ *can be solved in* $O(2^{3n/4})$ *time and* $O(2^{n/4})$ *space.*

Proof. For each of the $O(2^{n/4})$ problems $P_1 \in T_1$ and for each of the problems $P_2 \in T_2$ one uses the algorithm described for the problems P_3 and P_4 of the tables T_3 and T_4 respectively, thus resulting in an algorithm of time complexity $2^{3n/4}$ and space $2^{n/4}$. ■

2.1.2 Schroeppel & Shamir

Later Schroeppel and Shamir in [SS81] improved the previous result based on an observation on the problem's structure: One sees that the algorithm used, accesses the elements of the sorted supertables sequentially, and thus there is no need to store all the possible combinations simultaneously in memory. All that is needed to be done, is being able to generate the combinations quickly in a sorted order. To implement this idea the authors use two priority queues in the following way:

1. Q' stores pairs of problems from T_1 and T_2
2. Q'' stores pairs of problems from T_3 and T_4

Both queues enable arbitrary insertions and deletions in logarithmic time and produce the pair with the smallest $P_1 \oplus P_2$ and largest $P_3 \oplus P_4$ sum accessible in constant time respectively.

We present now the algorithm which provides the paper's main result:

Algorithm 2 Schroeppel - Shamir

Sort T_2 into increasing problem order
Sort T_4 into decreasing problem order
Insert into Q' all the pairs $(P_1, T_2.\text{first})$ for $P_1 \in T_1$
Insert into Q'' all the pairs $(P_3, T_4.\text{first})$ for $P_3 \in T_3$
while $Q' \neq \emptyset \vee Q'' \neq \emptyset$ **do**
  $(P_1, P_2) \leftarrow$pair with smallest $P_1 \oplus P_2$ sum in Q'
  $(P_3, P_4) \leftarrow$pair with largest $P_3 \oplus P_4$ sum in Q''
  if $S = P$ **then**
    return 1 {Indicating success}
  else if $S < P$ **then**
    delete (P_1, P_2) from Q'
    if the successor P_2' of P_2 in T_2 is defined **then**
      $Q' \leftarrow (P_1, P_2')$
    end if
  else if $S > P$ **then**
    delete (P_3, P_4) from Q''
    if the successor P_4' of P_4 in T_4 is defined **then**
      $Q'' \leftarrow (P_3, P_4')$
    end if
  end if
end while
return 0 {Indicating Failure}

Theorem 2.1.2.1. *The space complexity of this algorithm is* $O(2^{n/4})$

Proof. At each stage a $P_1 \in T_1$ can participate in at most one pair in Q', and a $P_3 \in T_3$ can participate in at most one pair in Q''. The space complexity of the priority queues in thus bounded by $O(|T_i|) = O(2^{n/4})$. ∎

Theorem 2.1.2.2. *The time complexity of this algorithm is* $O(2^{n/2})$

Proof. Each (P_1, P_2) pair can be deleted from Q' at most once, since it is never reinserted into Q'. Similarly (P_3, P_4) pair can be deleted from Q'' at most once. At each iteration one pair is deleted from Q' or Q'' and thus the number of iterations cannot exceed the number of possible pairs, which is $O(2^{n/2})$. ∎

2.1.3 Merkle-Damgård hash function construction

The Merkle-Damgård transform is a way to construct collision-resistant hash functions in practise. Using this methodology we can maintain the collision-resistance property, while at the same time being able to handle inputs of arbitrary lengths.

Let (Gen, h) be a fixed-length collision-resistant hash function family for inputs of lengths $2l(n)$ and with output length $l(n)$. Gen generates the key s, which is public and is used to specify a particular function h^s from the family. While *kgen* remains unchanged we construct a variable-length hash function (Gen, H) as follows:

Algorithm 3 The Merkle-Damgård transform

input: a key s and a string $x \in \{0,1\}^*$ of length $L < 2^{l(n)}$
Set $B := \lceil \frac{L}{l} \rceil$
Pad x with zeroes so its length is a multiple of l
Parse the padded result as the sequence of $l-$bit blocks $x_1, \ldots, x_B$.
Set $x_{B+1} := L$ $\{L$ is encoded using exactly l bits$\}$
$z_0 = 0^l$
for $i = 1 \ldots B + 1$ **do**
    compute $z_i = h^s(z_{i-1} \parallel x_i)$
end for
Output z_{B+1}

For more on the basics of hash functions we refer the reader to [Yeh08]

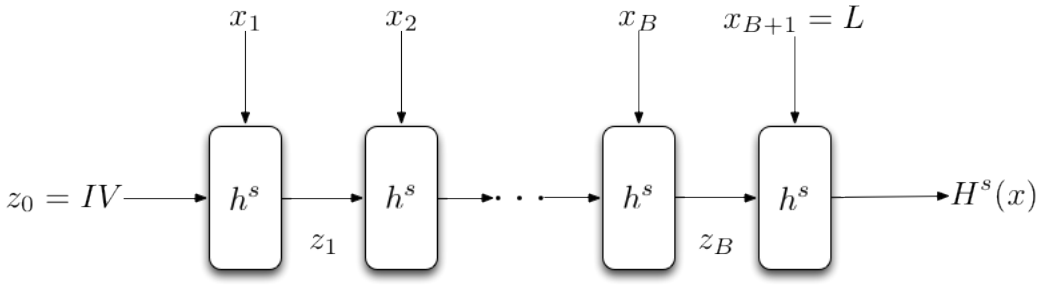


Figure 2.1: A pictorial approach of the Merkle-Damgård transform

2.2 Random Function Inversion

In [Hel80], Hellman proved that for every one-way permutation $f : [N] \to [N]$ and for every parameters S, T satisfying $S \cdot T = N$, there is a data structure of size $\widetilde{O}(S)$ and an

algorithm that, with the help of the data structure, given $f(x)$ is always able to find x in time $\widetilde{O}(T)$ [3] In particular any one-way permutation can be inverted in time $\sqrt{N}$ using $\sqrt{N}$ bits of advice. Hellman's algorithm only requires oracle access to the permutation. The construction works as follows:

- The adversary picks $\sqrt{N}$ points $x_1, \ldots, x_{\sqrt{N}}$, such that $x_{i+1} = f(x_i)$ and holds the start and end points in a table T.
- On a given y that the adversary wants to invert, he begins calculating $f(y), f(f(y)), \ldots$ until he reaches a point $f^j(y)$ which matches one (say x_i) of those to be found in T.
- Then in order to find y's inverse, the adversary starts calculating the values $f(x_i), f(f(x_i)), \ldots,$ until he reaches the point $f^m(x_i) = y$, which means that y's inverse will be $f^{m-1}(x_i)$.

In order to have a visual representation of the way the algorithm works, consider the graph induced by f, where the vertices are points from the function's domain and directed edges describe f's operation on a point. Then the above algorithm works perfectly in the case where this graph can be split in disjoint cycles (e.g. permutations) but only at these. Hellmans' breakthrough was the way that he extended this original idea to random functions as well. What he did was that proposing the following: Consider $l = N^{1/3}$ different functions $h_i : [N] \to [N]$, where $h_i(x)$ has the form $g_i(f(x))$ and g_i is a random function. Then for every function h_i, $1 \leq i \leq l$ take $x_{i_1}, x_{i_2}, \ldots, x_{i_l}$ points and store them in a table of size l. Let $t = N^{1/3}$; then each entry is a pair (x_{i_j}, u), where u is the t-th iterate of h_i on x_{i_j}. This table requires $O(m) = O(N^{1/3})$ space per function h_i and thus a total space of $O(ml) = O(N^{2/3})$ for all the $l = N^{1/3}$ tables. If f and the g_i are all independent random functions and $O(1)$ time computable, then f can be inverted at a random point with $\Omega(1)$ probability and in time $O(lt) = O(N^{2/3})$. Hellman suggest modifying the function by composing it with a fixed permutation of the input bits and to reason heuristically as if the new function behaved as an independently chosen new random function. Then the construction can be repeated and we get an algorithm of time and space complexity $N^{1/3}$ that inverts f in $N^{2/3}$ points. Iterating the process $N^{1/3}$ times provides an algorithm with $N^{2/3}$ complexity that inverts f everywhere.

Later Fiat and Naor in [FN99] make Hellman's argument rigorous by picking a good random hash function g, and then working with the new function $h(x) = g(f(x))$. If g were a truly random function and f were a function such that every output has few pre-images, then one can repeat Hellman's calculation that $N^{1/3}$ nearly disjoint paths of length $N^{1/3}$ exist.

[3] The notation $\widetilde{O}(\cdot)$ hides lower order factors that are polynomial in $logN$ but we will ignore such factors from now on in the interest of readability. We shall refer to S, the size of the pre-computed data structure (the advice) used by the algorithm, as the space used by the algorithm.

In [Rab10], De, Trevisan and Tulsiani introduce a new way to analyze the Fiat-Naor construction, which improves the complexity if one seeks to invert the given function in an ϵ fraction of the inputs. They also show that in an oracle setting it is not possible to do better than $\Omega(\sqrt{\epsilon}N)$. In their proof they use the adversarys' advice string to costruct a randomized Encoding/Decoding procedure, an idea which we use in proving the iHash PoSE 3.3 secure. Here we repeat the lemma proved in [Rab10], which we use in our proof.

Theorem 2.2.0.1. *Let $\mathcal{A}$ be an oracle algorithm, which makes at most T oracle queries and which takes an advice string of length S. Fix a parameter ϵ. There are randomized encoding and decoding procedures E and D, which use shared randomness and such that if f is a permutation and adv is an advice string such that*

$$\mathsf{Prob}[\mathcal{A}^f_{adv}(f(x)) = x] \geq \epsilon$$

then

$$\mathsf{Prob}[D(r, E(r, f)) = f] \geq .9$$

and the length of $E(R, f)$ is at most

$$\log N! - \frac{\epsilon N}{100T} + S + O(\log N)$$

Proof. Using the shared randomness r we generate a random subset $R \subseteq [N]$ such that each element of $[N]$ is independently chosen to be in R with probability $\frac{1}{10T}$

We call an element of R good if the following two properties hold:

1. $\mathcal{A}^f_{adv}(f(x)) = x$
2. for all oracle queries $q \in T$ we have that $q \in [N] \setminus R$ except possibly for the query x.

Let G be the set of all good elements and I be the set of elements, that $\mathcal{A}$ can invert. Then we have the following:

$$\mathsf{Prob}\left[\text{all queries of } \mathcal{A} \text{ are outside of R}\right] \geq \left(1 - \frac{1}{10T}\right)^T \approx 1 - e^{-10} \approx 1 - \frac{1}{100}$$

which means that

$$\mathsf{Prob}\left[\text{one query is inside of } R\right] = \frac{1}{100}$$

The events that $x \in R$ and that one query is in R are independent. Let K be the set of

elements $x \in R$ such that $\mathcal{A}$ inverts $f(x)$ but makes some queries inside of R. Then

$$E[|K|] = I \cdot \frac{1}{10T} \cdot \frac{1}{100} = \frac{I}{1000T}$$

By Markov's inequality we have that

$$\mathsf{Prob}\left[|K| \geq \frac{I}{50T}\right] \leq \frac{E(|K|)}{I/50T}$$

so

$$\mathsf{Prob}\left[|K| < \frac{I}{50T}\right] \geq 1 - \left(\frac{I}{1000T}\right) \cdot \frac{1}{I/50T} = .95 \tag{2.1}$$

By the Chernoff bound we have that

$$\mathsf{Prob}\left[|R| > \frac{I}{20T}\right] \geq .95 \tag{2.2}$$

By (2.1) and (2.2) for the set of good elements, G, we have that

$$|G| \geq |K| - |R| = \frac{3I}{100T}$$

Next we describe the Encoding and Decoding procedures, assuming that $|G| \geq \epsilon N/100T$.

The encoding contains the following information:

- The advice string adv
- The cardinality of the set G
- The set $f(R)$, encoded using $\log \binom{N}{|R|}$ bits
- The values of f restricted to $f : [N] \setminus R \to [N] \setminus f(R)$, encoded using $\log(N - |R|)!$ bits

  [4]
- The set $f(G)$ of the images of good elements of R, encoded using $\log \binom{|R|}{|G|}$ bits
- The values of f restricted to $f : R \setminus G \to f(R \setminus G)$, encoded using $\log(|R| - |G|)!$ bits

[4] That is, this part of the encoding is a permutation $g : [N] \setminus |R| \to N \setminus |R|$ with the meaning that if $g(i) = j$, then f maps the $i-$th element of the set $[N] \setminus R$ to the j-th element of the set $[N] - f(R)$. Note that knowledge of the sets R and $f(R)$ is needed to decode this part of the encoding. This will not be a problem because the decoder knows R, which is part of the common random string, and is given $f(R)$)

So choosing f a permutation, we are able to know exactly the amount of bits needed to represent f or any restriction of it. Given this we could replace f with any one to one and onto function.

The decoding proceeds as follows:

1. Initialize an empty table to store the values of f
2. Fill up the mapping from $[N] \setminus R$ to $[N] \setminus f(R)$.
3. For every element $y \in f(G)$ find its inverse. At this point we know the set G as well as the value of f on every point in $([N] \setminus R) \cup G$.
4. Compute f on $R \setminus G$. This can be done, since we now know G, R and the set $f(R \setminus G)$ as well as the permutation restricted to $R \setminus G$.

Based on the above the encoding's length is:

$$\begin{aligned} S + \text{ encoding of } f(R) + \text{ encoding of } f([N] \setminus R) + \text{ encoding of } f(G) + \text{ encoding of } f(R \setminus G) &= \\ S + \log\left(\frac{N!}{(N-|R|)!|R|!} \cdot (N-|R|)! \cdot \frac{|R|!}{(|R|-|G|!)|G|!} \cdot (|R|-|G|)!\right) + O(\log N) &= \\ S + \log N! - \log |G|! + O(\log N) & \end{aligned}$$

Theorem 2.2.0.2. *If $\mathcal{A}$ is an oracle algorithm that runs in time at most T and such that for every permutation f there is a data structure adv of size $\leq S$ such that*

$$\mathsf{Prob}[\mathcal{A}^f_{adv}(f(x)) = x] \geq \epsilon$$

then

$$S \cdot T = \Omega(\epsilon N)$$

■

Chapter 3

PoSEs Constructions

3.1 PoSE Preliminaries

A non interactive Proof of Secure Erasure (PoSE) is a protocol executed between a computationally powerful verifier $\mathcal{V}$ and a more limited prover $\mathcal{P}$. Here we are only interested in the case where the provers' limitation is only connected to his limited storage capacity. Both the verifier and the prover will have to perform a computation, that will need using so much storage as the provers' total memory capacity. The verifier will be able to perform the computation with relative ease, while the prover will have to exhaust all his memory. The relation between this protocol and general time/space tradeoffs is obvious: The prover has to use all his memory or else pay in computational cost. As an example one could picture the protocol running in a wireless sensor network: The prover would be any node of the network, while the verifier would be an external user, who either remotely or locally runs the protocol with that node.

3.1.1 Previous PoSEs

The easiest way to implement a Proof of Secure Erasure is the following:

1. The verifier, who knows beforehand the amount of memory that the prover has, sends to the prover a message as big as the prover's memory, that contains fresh randomness.
2. The prover stores the message and sends it back to the verifier.
3. The verifier compares the message he received to the message he sent and accepts if the two messages are identical.

The idea behind the scheme is sound but the communication complexity is high and renders it inefficient for most practical applications.

The authors in [PT10] improve on this initial idea, trading however its' unconditional security for lower communication complexity. In their scheme, again $\mathcal{V}$ sends a string to $\mathcal{P}$, as large as the latters' memory capacity, but now the prover has to calculate a MAC with key on that string, using the last k bits of the string as the key and then sending back the result of this calculation. The authors consider also the straightforward way of using a cryptographically suitable hash function (chosen for instance from the SHA family) but of course abandon it immediately as it yields an insecure protocol: Due to the functions' design (cf. (2.1.3)), a malicious prover does not need to store the entire message; he can start computing in real time, while receiving the message and no need for storing the random message would arise.

Although the communication complexity in this scheme is reduced, still the initial step of sending a message as large as the provers' memory leaves enough space for improvement. A solution that would solve the problem would have the verifier sending a very small string to the prover, who in his turn would "unravel" this string to a data structure so big, that he would have to use all of its' memory to hold it.

And this is exactly what is done in [DKW11], where Dziembowski et al. construct what they call an $(m-\delta, \epsilon)$ uncomputable function, which can be easily computed when m space is used, but when $m-\delta$ space is used it can be computed with at most a negligible probability ϵ. The prover has to create a graph, which he will have to keep in his memory during the whole time the protocol is ran. We describe this tool and how a PoSE can be formed using it, later in 3.4 once we have introduced the pebbling game on directed acyclic graphs.

Using this tool, the communication bandwidth is reduced dramatically but the protocols' time complexity increases from linear to quadratic, rendering the protocol practically unsuitable for most applications.

3.1.2 Security Definition

Before we describe the schemes we developed, it is necessary to define the security for an efficient Proof of Secure Erasure.

Definition 3.1.2.1. A PoSE is $(t, s, p)-$feasible if

$$\mathsf{Prob}\left[\begin{array}{c}\mathcal{V} \text{ accepts} \\ \mathcal{P} \text{ ran for } t \text{ time and used } s \text{ space}\end{array}\right] \geq p,$$

Definition 3.1.2.2. For an $0 < \alpha < 1$ a PoSE is α-robust if for every $0 < \gamma < 1$,and for every s there exists a $t > 0$ such that the protocol is $(t, s, 1-\gamma)$-feasible and $(t, \alpha s, \gamma)$-infeasible.

Definition 3.1.2.3. An α-robust PoSE is efficient if t is polynomial in $\log \gamma^{-1}, s,$ and $(1-\alpha)^{-1}$.

3.2 A first pass

From what we have described so far we could say that we are now in position to know what we need from a PoSE in order for it to be efficient: We would need that the verifier created an instance of a problem, which could be described with only a few bits. The answer to this problem should also be as long as only a few bits. This property would assure that the communication complexity would be as low as possible. In order also for the problem to be solved, the honest verifier should have to use all his available memory; and this should be done with acceptable time complexity (that is any time complexity less than quadratic in the size of the problem). Last but not least we should be sure, that any dishonest prover (that is one who would use any less than the honests' memory, would either be not be able to solve the problem at all, or in best case he would have to run in time at least quadratic on the inputs' size.

With these in mind we begin by describing our first attempts in proposing an efficient PoSE, which however were soon abandoned due to reasons that will hopefully become clear.

Knapsack PoSE: In this setting we would need the verifier to create a knapsack instance and a target value, and the prover would have to answer with a yes if and only if the target value could be generated by the instance given. However this idea cannot be easily exploited since for hard instances there is no proof that there does (or not) exist a better time-space tradeoff than a trivial exhaustive search.

Subset Sum PoSE: This is almost the same setting like the previous and has the same problems with it

Collision PoSE: In this setting the prover uses a hash function (which can be described with a few bits) and asks the prover to find a collision in it. However an adversary using Wagner's tree algorithm [Wag02] or Pollards' Rho algorithm could solve the problem using only little (or even constant) space.

What is common behind the problems that arose in trying to create a PoSE in the ways described just above, is the fact that although there is a great amount of work done on NP-hard problems in terms of their time complexity, most of them have not been studied in the average case and/or they have either not been studied in their space requirements or they have not been studied in a Time-Memory trade-off.

Combining however the problems mentioned above we were able to come up with a special case which lead to the invert-hash PoSE detailed in 3.3.

3.3 Invert-Hash PoSE

3.3.1 Adversarial Model

In this first attempt towards a better POSE we have refrained our adversary, in the sense that we consider the model, where he acts on two distinct phases. Namely a so-called "preparation" phase, during which he is allowed to perform any queries he wants, store the answers and perform any calculation he considers necessary. The results of this phase are stored in the adversarys' memory. In the second phase the adversary runs the protocol and uses the results stored in his memory as advice. In this phase he is not allowed to perform any new queries to the oracle.

3.3.2 PoSE description

In the setting described above we propose the following POSE:

1. $\mathcal{P}$ and $\mathcal{V}$ have the descriptions of three hash functions f_1, f_2 and f, with the property that for every $f(x)$ there exist x_1 and x_2 such that $f(x) = f(x_1) + f(x_2)$.
2. The verifier $\mathcal{V}$ chooses a challenge x and sends over to $\mathcal{P}$, $f(x)$
3. $\mathcal{P}$ inverts $f(x)$ and sends the result to $\mathcal{V}$.

In order to implement the above protocol, we will need the three above mentioned random functions with domains and co-domains as specified below:

$$f : [N^2] \to [N^2]$$
$$f_1 : [N] \to [N^2/2]$$
$$f_2 : [N] \to [N^2/2]$$

as well as a function

$$G : [N^2/2] \times [N^2/2] \to [N^2]$$

with the following property:

$$\forall x \in [N^2]\, \exists x_1, x_2 \in [N] \text{ such that}$$
$$G(f_1(x_1), f_2(x_2)) = f(x)$$

For a given $y \in [N^2]$ such that $f(x) = y$, in order to find it's pre-image x, we need to find x_1 and x_2, such that $x = x_1 + Nx_2$ and satisfy the following property:

$$f(x) = G(f_1(x_1), f_2(x_2))$$

Of course asking for the values x_1 and x_2 is actually asking for the inversion of the two random functions f_1 and f_2 on the given inputs and how one could do it efficiently is what we will be concerned in what follows. Of course one could perform an exhaustive search of the two functions' tables, but this is far from being time efficient while at the same time it can be done using only constant space. However one could use a time/space tradeoff technique like the ones briefly described in 2.2 and this is how we choose our prover to perform the task of random function inversion: We use the algorithm in [HS74] and achieve subquadratic time complexity while using the provers' all available memory. We do this by creating two tables T_1 and T_2 in which we insert the images $f_1(x_1)$ and $f_2(x_2)$ for all $x_1, x_2 \in [N]$ sorted in increasing and decreasing order respectively. Then we add the first element of the one table to the first of the second and check if this is the element we want to invert. If yes, then we look at the tables and return the pre-images of the two elements, that added to the element at hand. Else according to whether the sum we obtained before was greater or less than the target value, we proceed by adding to the element of the first (respectively the second) table that we had from the previous step, the next element of the second (respectively the first) table. The time complexity of this algorithm is governed by the sorting step, which will require $O(N/2 \log N/2)$ time and the space complexity will be exactly N. The algorithm is described in detail below.

Algorithm 4 invert-Hash

Input: y, T_1, T_2 {y is the value upon which which we want to invert f, T_1 holds the description of f_1 in form of pairs $(x, f_1(x))$ and T_2 respectively for f_2}
Sort T_1 in decreasing order on $T_1[1]$
Sort T_2 in increasing order on $T_2[1]$
while $T_1 \neq \emptyset \vee T_2 \neq \emptyset$ **do**
  $S \leftarrow G(T_1[1].\text{first}, T_2[1].\text{first})$
  if $S == y$ **then**
    $x_1 \leftarrow T_1[0].\text{first}$
    $x_2 \leftarrow T_2[0].\text{first}$
    return (x_1, x_2) {Solution found}
  end if
  if $S < y$ **then**
    delete T_1.first from T_1
  end if
  if $S > y$ **then**
    delete T_2.first from T_2
  end if
end while

Using the functions f, f_1, f_2 and G that satisfy the properties stated in 3.3.2 and s_1, s_2 as the necessary paddings for x_1 and x_2 in order for them to belong in the domains of f_1 and f_2 respectively, we get the following PoSE, which we call invert-Hash PoSE

1. $\mathcal{V}$ chooses x_1, x_2, s_1, s_2 such that

$$G(f_1(s_1||x_1), f_2(s_2||x_2)) = f(x_1 + x_2 N) = y$$

   and sends s_1, s_2 and y to $\mathcal{P}$

2. $\mathcal{P}$ inverts f on y by finding x_1 and x_2 and using algorithm (4) and sends $x = x_1 + x_2 N$ to $\mathcal{V}$, who accepts if and only if

$$f(x) = y$$

In the following section we analyze the security of our PoSE, assuming adversaries of the type described in 3.3.1.

3.3.3 Security Proof

In order to prove the security of this PoSE we first need to prove the following

Lemma 3.3.3.1 (Fact 1)**.** *Suppose there exists a randomized encoding procedure* $Enc : \{0,1\}^N \times \{0,1\}^r \to \{0,1\}^m$ *and a decoding procedure* $Dec : \{0,1\}^m \times \{0,1\}^r \to \{0,1\}^N$ *such that*

$$\mathsf{Prob}_{r \in U_r}[Dec(Enc(x,r),r) = x] \geq \delta$$

Then

$$m \geq N - \log 1/\delta$$

Proof. By a standard averaging argument, we get that there is an r such that for at least a δ fraction of the x' s, $Dec(Enc(x,r),r) = x$. However, that means that $Enc(x,r)$ must attain at least $\delta 2^N$ values as x varies over $\{0,1\}^N$. As the total number of values that $Enc(x,r)$ can take is bounded by 2^m, $2^m \leq \delta 2^N$, thus giving us the required inequality. ■

Lemma 3.3.3.2. *Let* $\mathcal{A}$ *be an algorithm, that succeeds in inverting a given element* $y \in [N^2]$ *with*

$$\mathsf{Prob}[\mathcal{A}(y) = f^{-1}(y)] \geq \delta$$

Then there exist algorithms $\mathcal{A}_1, \mathcal{A}_2$ *that succeed in inverting* $f_1(y_1)$ *and* $f_2(y_2)$ *and*

$$\mathsf{Prob}[\mathcal{A}_1(y_1) f_1^{-1}(y_1)] = \mathsf{Prob}[\mathcal{A}_2(y_2) = f_2^{-1}(y_2)] \geq \delta$$

Proof. We describe the algorithm $\mathcal{A}_1$, that inverts an element y_1 using $\mathcal{A}$ and works as follows:

1. $\mathcal{A}_1$ chooses an element $x_2 \in [N]$, forms $y = y_1 + f_2(x_2)$ and passes it to $\mathcal{A}$.
2. $\mathcal{A}$ returns all (x_1, x_2), such that $f_1(x_1) + f_2(x_2) = y$.
3. $\mathcal{A}_1$ finds (x_1, x_2) and returns x_1.

We will show that $\mathcal{A}_1$ succeeds in inverting a given element y_1 with probability at least δ. Define

$$\begin{aligned}
&\Psi_1 = \{y_1 \in [N^2/2] : \exists y \in [N^2] \exists (x_1,x_2) \in [N]^2 : (x_1,x_2) \in \mathcal{A}(y) \wedge f_1(x_1) = y_1\},\\
&\Psi_2 = \{y_2 \in [N^2/2] : \exists y \in [N^2] \exists (x_1,x_2) \in [N]^2 : (x_1,x_2) \in \mathcal{A}(y) \wedge f_2(x_2) = y_2\},\\
&S = \{(x_1,x_2) \in [N]^2 : \exists y \in [N^2] : (x_1,x_2) \in \mathcal{A}(y)\},\\
&S^{f_1,f_2} = \{(y_1,y_2) \in [N^2/2]^2 : \exists (x_1,x_2) \in S : f_1(x_1) = y_1 \wedge f_2(x_2) = y_2\}
\end{aligned}$$

Now let $(y_1, y_2) \in S^{f_1,f_2}$. Then there exists $(x_1, x_2) \in S$ such that $f_1(x_1) = y_1$ and $f_2(x_2) = y_2$. Since $(x_1, x_2) \in S$, there exists a y such that $(x_1, x_2) \in \mathcal{A}(y)$. So for this y, we have that $y_1 \in \Psi_1$ and $y_2 \in \Psi_2$, which means that $S^{f_1,f_2} \subseteq \Psi_1 \times \Psi_2$. Since $|S| =$

$|S^{f_1,f_2}|$ (because f_1 and f_2 are bijections) and by the hypothesis $|S| \geq \delta N^2$, we have that $|\Psi_1 \times \Psi_2| \geq \delta N^2$.

Since $S \neq \emptyset$, we have that $S^{f_1,f_2} \neq \emptyset$ and therefore $\Psi_1 \neq \emptyset$ and $\Psi_2 \neq \emptyset$. Observe also that since f_1 and f_2 are bijections, we have that $|\Psi_1| \leq N$ and $|\Psi_2| \leq N$. Assume that $|\Psi_1| < \delta N$ and $|\Psi_2| = N$. Then $|\Psi_1 \times \Psi_2| < \delta N^2$, which is a contradiction. Therefore it must hold that $|\Psi_1| \geq \delta N$ and $|\Psi_2| \geq \delta N$.

To conclude the proof, let y_1 be a random element from $f_1([N])$. Since $\mathcal{A}_1$ inverts all the elements in Ψ_1 and $|f_1([N])| = N$, we have that

$$\mathsf{Prob}[\mathcal{A}_1(y_1) = f^{-1}(y_1)] \geq \frac{\delta N}{N} = \delta$$

■

Lemma 3.3.3.3 (Enc/Dec description)**.** *Let $\mathcal{A}$, be a probabilistic poly time algorithm, that on input (α, y), where y is the element to be inverted and α is an advice string of length $|\alpha| = \epsilon N$, returns the set $\{(x_1^j, x_2^j) : y = f_1(x_1^j) + f_2(x_2^j), j = 1, \ldots, k, k \leq N\}$ of all the preimages of y, with the property that $(f_1(x_1^j) < f_1(x_1^{j+1}))$ and $(f_2(x_1^j) > f_2(x_1^{j+1}))$ and let $\mathcal{A}$ succeed on inverting with probability*

$$\mathsf{Prob}[\mathcal{A}(\alpha, f(x)) = x] \geq \delta$$

Then using $\mathcal{A}$ we can produce a randomized encoding procedure for f_1 and f_2.

Proof. We begin by describing the Encoding and Decoding procedures:

Encoding Let $|\alpha|$ be the length of the advice string generated by f_1 and f_2. The encoding consists of the advice string α and a table T, which contains the $2(1-\delta)N$ elements, not inverted by $\mathcal{A}$.

Decoding

1. Initialize a table T' that will hold the values of f_1 and f_2
2. Fill the T' with the contents of T
3. For every element in T' that has not yet been inverted, use $\mathcal{A}$ to invert it.

Next we calculate the space needed for the encoding:

- Encode the values of $f_1 : [N] \to [N^2/2]$ that $\mathcal{A}$ cannot invert, using $\log (1-\delta)N!$ bits.
- Encode the set $f_1((1-\delta)N)$ of the images of the elements, that cannot be inverted by $\mathcal{A}$ using $\log \binom{N^2/2}{(1-\delta)N}$ bits.

The total space needed for the encoding is:

$$
\begin{aligned}
|\alpha| + 2\log((1-\delta)N)! + 2\log\binom{N^2/2}{(1-\delta)N} &= |\alpha| + 2\log((1-\delta)N)!\binom{N^2/2}{(1-\delta)N} \\
&= |\alpha| + 2\log\frac{((1-\delta)N)!\frac{N^2}{2}!}{(1-\delta)N!(\frac{N^2}{2}-(1-\delta)N)!} \\
&= |\alpha| + 2\log\frac{(N^2/2)!}{(N^2/2-(1-\delta)N)!} \\
&= |\alpha| + 2\log\prod_{k=1}^{(1-\delta)N}\frac{N^2}{2} - (1-\delta)N + k \\
&= |\alpha| + 2\log\prod_{k=1}^{(1-\delta)N} N^2\left(\frac{1}{2} - \frac{1-\delta}{N} + 2\frac{k}{N^2}\right) \\
&= |\alpha| + 2\log N^{2(1-\delta)N}\prod_{k=1}^{(1-\delta)N}\left(\frac{1}{2} - \frac{1-\delta}{N} + 2\frac{k}{N^2}\right) \\
&= |\alpha| + 2\log N^{2(1-\delta)N} + 2\log\prod_{k=1}^{(1-\delta)N}\left(\frac{1}{2} - \frac{1-\delta}{N} + \frac{k}{N^2}\right) \\
&= |\alpha| + 4(1-\delta)N\log N + 2\log\prod_{k=1}^{(1-\delta)N}\left(\frac{1}{2} - \frac{1-\delta}{N} + \frac{k}{N^2}\right) \\
&\leq |\alpha| + 4(1-\delta)N\log N
\end{aligned}
$$

■

Proposition 3.3.3.1. *The Invert-Hash-PoSE is α-robust.*

Proof. Let S be the memory size that we want to securely erase, γ the adversary's success probability and αS the amount of memory, that the adversary will use. It is easy to see, that using the algorithm described in 4 the Invert-Hash-PoSE is $(O(S\log S), S, 1-\gamma)$-feasible, since it needs $O(S\log S)$ time in order to sort the elements and needs S space to hold them. Since in order to find the inverse element, it has to go through all the table that contains f_1 and f_2, we see that

$$
\mathsf{Prob}\left[\begin{array}{c}\mathcal{V}\text{ accepts} \\ \mathcal{P}\text{ ran for }O(S\log S)\text{ time and used }S\text{ space}\end{array}\right] = 1,
$$

Next we show that Invert-Hash-PoSE is $(O(S\log S), \alpha S, \gamma)$ infeasible. Assume that it

is $(O(S \log S), \alpha S, \gamma)$ feasible. Then

$$\mathsf{Prob}\left[\begin{array}{c} \mathcal{V} \text{ accepts} \\ \mathcal{P} \text{ ran for } O(S \log S) \text{ time and used } \alpha S \text{ space} \end{array}\right] \geq \gamma$$

In other words there exists an algorithm $\mathcal{A}$ such that given an advice string of length αS and an element y to invert succeeds with probability $\geq \gamma$.By lemma 3.3.3.3 and using $\mathcal{A}$ we can construct a randomized encoding procedure that uses space $\alpha S + 4(1-\gamma)S \log S$ bits. By lemma 3.3.3.1 we have that

$$\begin{aligned}
\alpha S + 4(1-\gamma)S \log S &\geq \log \left(\frac{S^2}{2}\right)_S \\
\alpha S + 4(1-\gamma)S \log S &\geq \log \left(\frac{S^2}{2}\right)_S \\
&\geq \log \left(\frac{S^2}{2} - S\right)^S \\
&= S \log \left(\frac{S^2}{2} - S\right) \\
4(1-\gamma) \log S &\geq \log \left(\frac{S^2}{2} - S\right) - \alpha \\
&\geq (1-\alpha) \log \left(\frac{S^2}{2} - S\right) \\
(1-\gamma) &\geq (1-\alpha) \frac{\log \left(\frac{S^2}{2} - S\right)}{4 \log S} \\
\gamma &\leq 1 - (1-\alpha)\frac{1}{2} \\
\gamma &\leq 1 - (1-\alpha)\frac{1}{2}
\end{aligned}$$

Suppose that the protocol is repeated k times. Then given that

$$\left(1 - (1-\alpha)\frac{1}{2}\right)^k \leq e^{-(1-\alpha)\frac{1}{2} \cdot k}$$

we have that

$$\begin{aligned} e^{-(1-\alpha)\frac{1}{2}\cdot k} &\leq \gamma \\ -(1-\alpha)\cdot k/2 &\leq \ln\gamma \\ (1-\alpha)\cdot k &\geq 2\ln\gamma^{-1} \\ k &\geq \frac{2\ln\gamma^{-1}}{(1-\alpha)} \end{aligned}$$

which means that

$$k = \Omega(1.39(1-\alpha)^{-1}\log\gamma^{-1})$$

■

3.4 Graph based PoSE

3.4.1 Introduction

From what we have seen up to now, it is clear that for a PoSE to be practical, a computational problem must be used, for which we know lower bounds in its space complexity and the time needed for the honest prover to solve the problem must be less than quadratic. Not many problems have been studied in both their space and time requirements and from the ones studied in such a fashion, we have not been able to come up with a better scheme than the one described in 3.3. There is however a problem which has been studied extensively and combines all of our needs in order to come up with a practical Proof of Secure Erasure; namely the problem of pebbling games over graphs. We refer to [Nor11] for a detailed survey - here we provide only the necessary background for our purposes. We recall first that for a DAG G, a sink vertex is a vertex with out-degree 0 and a source vertex is a vertex with in-degree 0.

Definition 3.4.1.1. (Pebble game). Let G be a directed acyclic graph (DAG). A pebble game on G is the following one-player game. At any time t, we have a configuration $\mathbb{P}_t$ of pebbles on the vertices of G, at most one pebble per vertex. The rules of the game are as follows:

1. If all immediate predecessors of an empty vertex v have pebbles on them, a pebble may be placed on v. In particular, a pebble can always be placed on a source vertex.
2. A pebble may be removed from any vertex at any time.

A pebbling of G, also called a pebbling strategy for G, is a sequence of pebble configurations $\mathbb{P} = \langle \mathbb{P}_0, \ldots, \mathbb{P}_\tau \rangle$. In case $\mathbb{P}_0 = X$ and $\mathbb{P}_\tau = Y$ where X, Y are the source and sink vertices of G then we call this a complete strategy. Furthermore, if for all $t \in [\tau]$, $\mathbb{P}_t$ follows from $\mathbb{P}_{t-1}$ adhering to the rules above, we call this a legal strategy. The time of a pebbling $\mathbb{P} = \{\mathbb{P}_0, \ldots, \mathbb{P}_\tau\}$ is simply $time(\mathbb{P}) = \tau$ and the space is $space(\mathbb{P}) = \max_{0 \le t \le \tau}\{|B_t|\}$ where B_t represents the set of vertices that carry a pebble in time t. The pebbling price of G, denoted $Peb(G)$, is the minimum space of any complete legal strategy for G.

In the case of black-white pebbling, one has in his disposal two sets of pebbles, black and white, where the black can be placed following the rules mentioned above, while the white can be placed following the rules stated here:

1. A white pebble can be placed on any empty vertex at any time.
2. If all immediate predecessors of a white-pebbled vertex u have pebbles on them, the white pebble on u can be removed. A white pebble can always be removed from a source vertex.

Definition 3.4.1.2. (Compact Recursive Representation) A family of Directed Acyclic Graphs $\{G_n\}_{n=1}^{\infty}$ will be called Compactly Recursively Representable (CRR), if for every graph $G_n = (V_n, E_n)$, there exists a circuit C of polylogarithmic size in $|V_n|$ that given a node returns its incoming edges.

In other words, a graph that belongs in this family, will have a short (independent of the usual matrix or list) representation, which is what we need in our protocol, since for the verifier to send the whole graph in its matrix or list representation, would increase the communication complexity to the levels of [PT10].

3.4.2 Computational model

Consider $G = (V, E)$ to be a DAG of $|V| = n$ vertices. In G we distinguish some special vertices : an *input vertex* is a vertex with no incoming edges and an *output vertex* is a vertex that has no outgoing edges.

For any such graph G we define the following symbolic labeling : input vertices are labeled by x_i where i ranges over $\{1, \ldots, k\}$ where k is the number of input vertices. Any other node v is labeled by $H(v, l_1, \ldots, l_m)$ where m is the number of incoming edges and $l_1, \ldots, l_m$ are the corresponding labels of the incoming (parent) vertices.

Definition 3.4.2.1. Let $U = \{0,1\}^s$. Given a DAG $G = (V, E)$ with k input vertices, m output vertices and bounded in-degree d, we define for a given function $H : V \times U^d \to U$ the function $G^H : U^k \to U^m$ to be the function that maps $x_1, \ldots, x_k \in U$ to the values $y_1, \ldots, y_m \in U$ that correspond to the evaluations of the symbolic labelings of the output vertices of G using the function H.

We now fit the notion of ex-post-facto pebbling from [DKW11] to our setting.

Definition 3.4.2.2. Fix $U = \{0,1\}^s$ and DAG G. Given a probabilistic algorithm A utilizing an oracle $H : V \times U^s \to U$. The ex-post-facto pebbling of G corresponding to A is a pebbling strategy $\mathbb{P}$ parameterized by input $x_1, \ldots, x_k \in U$, the choice of H and the coins ρ of A that satisfies the following :

1. The time of the pebbling equals the number q of oracle queries of A to H.
2. Initially pebbles are placed on all k of the input vertices of G.
3. If in the i-th query A asks H for an input u and it holds that u equals the label of a node v then a pebble is placed on that vertex for the configuration $\mathbb{P}_i$.
4. Suppose a label l of a non-output vertex v never appears in any query j succeeding the i-th query until the sequence terminates or the label of v is recomputed. Then, no pebble is placed on v in any configuration $\mathbb{P}_{i+1}, \ldots, \mathbb{P}_j$.

We say that an algorithm A has an execution that computes the function G^H for a given function H if it happens that the ex-post-facto pebbling of G corresponding to A on input $x_1, \ldots, x_k$ for some coins ρ of A is complete (i.e., it terminates in a configuration where all output nodes of G are pebbled). We prove the following theorem for the case when the function H behaves as a random oracle.

Theorem 3.4.2.1. *Fix $U = \{0,1\}^s$ and a d-bounded DAG $G = (V, E)$ with $n = |V|$ and k input vertices. Consider an algorithm A and the random variable $\mathbb{P}$ defined as the ex-post-facto pebbling of G parameterized by $x_1, \ldots, x_k$ selected uniformly at random from U and H selected uniformly at random from $(V \times U^d \to U)$. Then we have that*

$$Pr[\mathbb{P} \text{ is legal}] \geq 1 - q \cdot 2^{-s}$$

Proof. Let $\mathbb{P} = \langle \mathbb{P}_0, \ldots, \mathbb{P}_q \rangle$ be a possible pebbling strategy of A. If $\mathbb{P}$ is not legal this means that there exists a smallest $i \geq 1$ such that the transition from $\mathbb{P}_{i-1}$ to $\mathbb{P}_i$ does not adhere to the rules of definition **??**. This means that for some $i \in \{1, \ldots, q\}$, a pebble appears for vertex v in $\mathbb{P}_i$ while in $\mathbb{P}_{i-1}$ at least one of the parent nodes of v have no pebble placed in them. This means that the label of the parent node of v was never placed. In either case the fact that the ex-post-facto pebbling places a pebble on v means that the i-th query to H by A contains the label l of the node v'. We first consider case (2). We know that in this case v is not an input node (given that input nodes have pebbles placed on them automatically in any ex-post-facto pebbling). Let $a = (l'_1, \ldots, l'_d, v)$ be the string that defines the label $l = H(a)$. Since there was never a pebble in v' this means that a was never queried. It follows that the value l is uniformly random over U from the perspective of A, hence predicting it correctly as part of the i-th query can occur with probability at most 2^{-s}. The result follows given that i ranges in $\{1, \ldots, q\}$. ∎

Using the above theorem we conclude that any algorithm A performs legal peblings. Next we need to establish a relation between the price of pebbling and the space required by an algorithm A. We have the following:

Theorem 3.4.2.2. *Fix $U = \{0,1\}^s$, a d-bounded DAG $G = (V, E)$ with $n = |V|$ and k input vertices, and $H : V \times U^d \to U$. (1) There exists an algorithm $A^{H,P}$ in the RAM model that on input $x_1, \ldots, x_k$ computes $G^H(x_1, \ldots, x_k)$ using $Peb(G)$ space and time $O(|V|)$ where $P : V \to V^d \cup \{\perp\}$ is a function that returns the d parents of a given node v or $\perp$ if it is a source node. (2) Any other algorithm A that agrees with $G^H(x_1, \ldots, x_k)$ on a fraction of inputs above α uses space $Peb(G)$.*

Proof. (1) The proof is using P to perform a depth first search over G. Further details are omitted.

(2) Consider an execution of A for which the space complexity is less than $Peb(G)$. This means that there is a strategy that calculates $G^H(x_1, \ldots, x_k)$ but the pebbling induced by A in this execution path has price strictly smaller than $Peb(G)$ something that suggests that either it is not complete or not legal (given $Peb(G)$ is the minimum such price for complete and legal strategies). Given that there is an α fraction of complete strategies there should be at least $\alpha - q2^{-s}$ that are legal. ■

Having described the computational model upon which we will be relying, we next describe the graph family that will be used in the construction of our PoSE. An essential role as the building blocks of these graphs will be played by the superconcentrator graph family, which we describe next giving also some well known results about it, which will help us prove the security of our PoSE.

3.4.3 Superconcentrators

Superconcentrators are graphs that solve the problem of connecting N clients to N servers in a setting where either the clients or the servers are interchangeable and therefore it does not matter which client is connected to which server. A formal definition follows:

Definition 3.4.3.1. A directed acyclic graph G with N input and N output nodes, will be called an $N-$ superconcentrator if for every $r \leq N$, every set of r inputs, and every set of r outputs, there exists an $r-$flow (a set of r vertex-disjoint directed paths) from the given inputs to the given outputs.

As a first example of superconcentrator one could see that a graph family that satisfies the above definition is the complete bipartite graph with N nodes. However we cannot apply this family in the construction of a PoSE, since in our computational model the node indegree must be bounded.

A first non-explicit and low density [1] construction of superconcentrators was given by Pippenger in [Pip77]. Currently the record is held by Schöning [Sch06]. Regarding the explicit construction on the other hand, the most recent advances were made in [AC03], a construction of an $N-$superconcentrator with $44N + O(1)$ edges and $N = k \cdot 2^l$ with $k = 262,080$. For our purposes, the graphs produced must belong in the CRR family of graphs 3.4.1.2, and indeed the superconcentrators proposed by Alon do fall into this family, a result that we prove in the appendix B.0.4.2.

Next we prove some basic results on superconcentrators, which we will use in our protocol constructions.

[1] where graph density is defined as $2|E|/(|V|(|V|-1))$

Lemma 3.4.3.1. *Suppose that $Q : u \rightsquigarrow v$ is a path in G and that $\mathbb{P} = \{\mathbb{P}_\sigma, \mathbb{P}_{\sigma+1}, ..., \mathbb{P}_t\}$ is a black-white pebbling such that the whole path Q is completely free of pebbles at times σ and t but the endpoint v is pebbled at some point in the time interval (σ, t). Then the starting point u is pebbled during (σ, t) as well.*

Proof. By induction over the length of the path Q. The base case $u = v$ is trivial. For the induction step, let w be the immediate successor of u on Q. By the induction hypothesis, w is pebbled and unpebbled again sometime during (σ, t). Then u must be covered by a pebble either when the pebble on w is placed there (if this pebble is black) or when it is removed (if it is white). The lemma follows. ■

Lemma 3.4.3.2. *Let G be an $N-$ superconcentrator, S the set of its sources and Z the set of its sinks. Then for every pebble configuration $\mathbb{P}$ with $Space(\mathbb{P}) < s$ there exist $S' \subseteq S$ and $Z' \subseteq Z$, with $|S'| \geq N - s$ and $|Z'| > s$ such that for every $s \in S'$ and $z \in Z'$ the vertex path from s to z is completely pebble free.*

Proof. Let $\mathbb{P}$ be a pebbling configuration, using space less than s and $S'' \subseteq S$ such that $|S''| = s + 1$ and $Z' \subseteq Z$, $|Z'| > s$. Since G is a superconcentrator we have that there exist $s+1$ vertex disjoint paths from S'' to Z'. Since $Space(\mathbb{P}) < s$ we have that at most s paths will be blocked by pebbles. Therefore there exist at least $N - s$ sources with completely pebble-free paths to Z'. Setting $S' = S \setminus S''$ we attain the result. ■

Lemma 3.4.3.3 (Basic Lower Bound Argument)**.** *Suppose that $\mathcal{P} = \{\mathbb{P}_\sigma, \mathbb{P}_{\sigma_1}, \ldots, \mathbb{P}_t\}$ is a conditional (i.e.$\mathbb{P}_\sigma \neq \emptyset$) black-white pebbling of an $N-$ superconcentrator such that $space(\mathbb{P}_\sigma) \leq s_\sigma$, $space(\mathbb{P}_t) \leq s_t$, and $\mathcal{P}$ pebbles at least $s_\sigma + s_t + 1$ sinks during the closed time interval $[\sigma, t]$. Then $\mathcal{P}$ pebbles and unpebbles at least $N - s_\sigma - s_t$ different sources during the open time interval (σ, t).*

Proof. For the pebbling configuration $\mathbb{P}_\sigma$ we have from lemma 3.4.3.2 that there exists a set $S' \subseteq S$ and a set $Z' \subseteq Z$, with $|S'| \geq N - s_\sigma$, $Z' > s_\sigma$ such that all paths from S' to Z are completely pebble free. Similarly for the configuration $\mathbb{P}_t$ there exists a set $S'' \subseteq S$ and a set $Z'' \subseteq Z$, with $|S''| \geq N - s_\sigma$, $Z' > s_\sigma$ such that all paths from S' to Z are completely pebble free. For the sets $S' \cup S''$, $Z' \cup Z''$ it holds that $|S' \cup S''| \geq N - s_\sigma - s_t$ and $|Z' \cup Z''| > s_\sigma + s_t$ and there exist completely pebble free paths from $S' \cup S''|$ to $Z' \cup Z''$. By lemma 3.4.3.1 it follows that since $\mathcal{P}$ pebbles the sinks $Z' \cup Z''$ then the sources in $S' \cup S''$ must also be pebbled in the interval (σ, t). ■

Theorem 3.4.3.1 (Pebble lower bound). *Any complete black-white pebbling of an $N-$ superconcentrator G in space at most s has to pebble at least $\Omega(N^2/s)$ sources.*

Proof. Let $\mathcal{P}_1 = \{\mathbb{P}^1_\sigma, \mathbb{P}^1_{\sigma+1}, \ldots, \mathbb{P}^1_\tau\}$ be a pebbling such that $Space(\mathcal{P}_1) \leq Space(\mathbb{P}_\sigma + \mathbb{P}_\tau) \leq s$. By lemma 3.4.3.3 we have that $\mathcal{P}$ pebbles at least s sinks, while pebbling and unpebbling at least $N - s$ sources. For the remaining $N - s$ sinks we can find another pebbling $\mathcal{P}_2 = \{\mathbb{P}^2_\sigma, \mathbb{P}^2_{\sigma+1}, \ldots, \mathbb{P}^2_\tau\}$ again using space at most s and apply the same reasoning as described above.

Repeating the procedure N/s times, we will have pebbled all the sinks of G in time $\Omega(N^2/s)$. ∎

Paul, Tarjan and Celoni in [PTC76] create a family of graphs for which we can find the lower bound pebbling price. This graph family construction we describe here as presented in [Nor11]. Graphs that belong in this family we will be calling Paul-Tarjan-Celoni (PTC) graphs.

Definition 3.4.3.2. Let $C(k) = SC_{N(k)}$ for $k = 0, 1, 2, \ldots$ denote any arbitrary but fixed family of superconcentrators with $N(k) = K \cdot 2^k$ sources and sinks for some constant $K \in \mathbb{N}^+$ and $\Theta(N(k))$ vertices of indegree 2. Then the PTC graph $\Xi(0)$ is $C(0)$, and $\Xi(i+1)$ for $i \geq 0$ is defined inductively as follows:

The graph $\Xi(i+1)$ has sources $s_{i+1}[j]$ and sinks $z_{i+1}[j]$ for $j = 1, 2, \ldots, N(i+1)$. It contains two copies $\Xi_1(i), \Xi_2(i)$ of the PTC graph of one size smaller with sources $s^c_i[j]$ and sinks $z^c_i[j]$ for $j = 1, 2, \ldots, N(i)$ and $c = 1, 2$, and two superconcentrator copies $C_1(i), C_2(i)$ with sources $x^c_i[j]$ and sinks $y^c_i[j]$ for $j = 1, 2, \ldots, N(i)$ and $c = 1, 2$. The edges in $\Xi(i+1)$ are all internal edges within $\Xi_1(i), \Xi_2(i)$ and $C_1(i), C_2(i)$, as well as the following edges:

1. $(s_{i+1}[j], x^1_i[j])$ and $s_{i+1}[j+N(i)], x^1_i[j])$ for $j = 1, \ldots, N(i)$, from the sources in $\Xi(i+1)$ to the sources of $C_1(i)$,
2. $(y^1_i[j], s^1_i[j])$ for $j = 1, \ldots, N(i)$, from the sinks of $C_1(i)$ to the sources of $\Xi_1(i)$,
3. $(z^1_i[j], s^2_i[j])$ for $j = 1, \ldots, N(i)$, from the sinks of $\Xi_1(i)$ to the sources of $\Xi_2(i)$,
4. $(z^2_i[j], x^2_i[j])$ for $j = 1, \ldots, N(i)$, from the sinks of $\Xi_2(i)$ to the sources of $C_2(i)$,
5. $(y^2_i[j], z_{i+1}[j])$ and $(y^2_i[j], z_{i+1}[j + N(i)])$ for $j = 1, \ldots, N(i)$, from the sinks of $C_2(i)$ to the sources of $\Xi_1(i+1)$,
6. $(s_{i+1}[j], z_{i+1}[j])$ for $j = 1, \ldots, N(i+1)$, directly form the sources to the sinks of $\Xi(i+1)$.

The question though remains, as to which should be the superconcentrator family $SC(i)$ used. In the PoSE we construct, we use the butterfly superconcentrator family (for more on butterfly graphs, cf.[Nor11]).

In the following image we show the butterfly graph with 8 in- output nodes and the PTC graph with 8 in- output nodes (PTC(8)) using as the superconcentrator family, the Butterfly graph one. In fact in our graph PoSE construction we will be using this superconcentrator family, since it is easy to see that it belongs to the CRR family of graphs. Also in the proof of the main lower bound lemma **??** we will be using the PTC(8) graph from **??**, in order to visualize the various cases that we will examine.

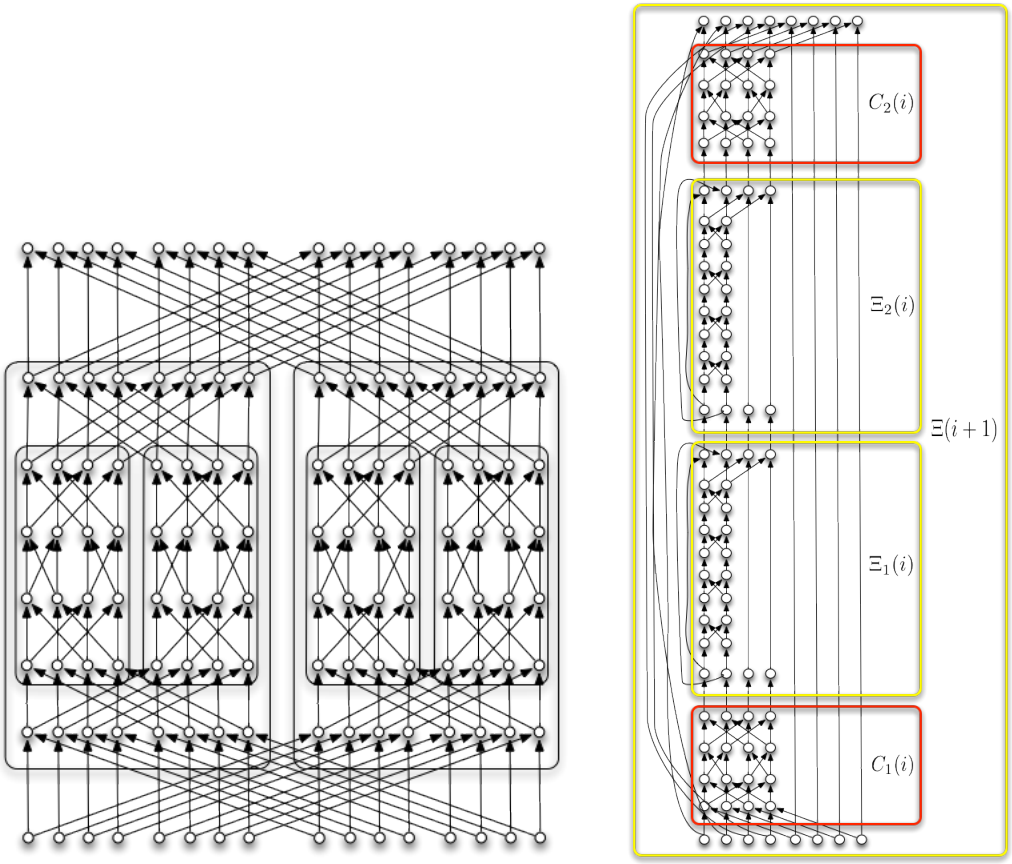


Table 3.1: Butterfly graph with 16 sinks and PTC graph with 8 sinks

For this graph family we have the following lemma, which is the main lemma upon which we base the result regarding the graph-based PoSE.

Lemma 3.4.3.4 (Lower Bound on Superconcentrator pebbling)**.** *Let* $m(i) = |S(i)| = |T(i)| = 2^i$*, let* $\Xi(i)$ *be a DAG as in the [PTC76] construction and suppose that in the interval* $[0, t]$ *at least* $c_1 m(i)$ *sinks of* $\Xi(i)$ *are pebbled with any colors in any order. Suppose also that at times* 0 *and* t *there are at most* $c_2 m(i)$ *pebbles on the graph. Then there is a time interval* $[t_1, t_2] \subseteq [0, t]$ *during which at least* $c_3 m(i)$ *sources of* $\Xi(i)$ *are pebbled and at least* $c_4 m(i)$ *pebbles are always on the graph.*

The proof will be carried out by induction. However we must be careful on how we will use the induction hypothesis, since in order to do so, one would need an upper bound of $c_2m(i)$ pebbles on the number of pebbles at the start and at the end of the subpebblings, but the lemma's statement only provides a $c_2m(i+1) = 2c_2m_i$ upper bound on the number of pebbles. What we want to do is apply the induction hypothesis on the two copies $\Xi_1(i)$ and $\Xi_2(i)$. In doing so we will come up against specific cases of pebbling configurations for which we will prove the lemma without calling the induction hypothesis. In proving these special cases, as well as in the core case of the lemmas' proof, certain demands will arise, which we will write down, as the further study of them determines the hidden constants in the asymptotic notation.

We now proceed to the description and proof of the special cases.

Special Case 1: Let $\mathcal{P}$ be a pebbling of $\Xi(i+1)$ meeting the prerequisites of the lemma 3.4.3.4. Suppose that there exists a time interval $[t_1, t_2] \subseteq [0, t]$ such that at least $c_3/2m(i)$ sources of the subgraph $\Xi_1(i)$ are pebbled and there are at least $c_2m(i)$ pebbles on $\Xi(i+1)$ throughout the whole interval. Then there exists an interval $[t_0, t_2] \subseteq [0, t]$ such that at least $c_3/2m(i)$ sources of $\Xi(i+1)$ are pebbled and at least $c_4m(i+1)$ pebbles are constantly on the graph.

Proof. Let $H_L^{'}$ be the subgraph induced by $C_1(i)$ along with the sources of $\Xi_1(i)$ and the left-hand half of the sources of $\Xi(i+1)$, i.e. the verices $\{s_{i+1}[j], s_i^1[j] : j \in [N(i)]\}$ along with the edges $\{(s_{i+1}[j], x_i^1[j]), j \in [N(i)]\}$, that are connected to the sources of $C_1(i)$ and the edges that come from the sinks of $C_1(i)$, $\{(y_i^1[j], s_i^1[j]), j \in [N(i)]\}$. In the same manner define $H_R^{'}$, that instead takes the right-hand half of the sources of $\Xi(i+1)$. For a pictorial approach to these two induced graphs, refer to 3.2. For these two graphs, it is easy to see that they are superconcentrators. Now let t_0 be the last time before t_1 at which there are no more than $c_2m(i+1)$ pebbles on the graph and assume that

$$\frac{c_3}{2}m(i) \geq c_2m(i+1) + 1 \tag{3.1}$$

Then by 3.4.3.3 there are at least $2(m(i) - c_2m(i+1) = (1-2c_2)m(i+1)$ (adding the ones from $H_L^{'}$ and $H_R^{'}$) sources of $\Xi(i+1)$ connected to pebble-free paths to the $c_3m(i)/2$, that are pebbled from t_1 to t_2. Then during the interval $[t_0, t_2]$ at least these sources of $G(i+1)$ must be pebbled and at least $c_2m(i) - 1$ pebbles must be constantly on the graph. Then assuming that

$$1 - 2c_2 \geq c_3 \tag{3.2}$$

and

$$c_2 m(i) - 1 \geq c_4 m(i+1) \tag{3.3}$$

and observing that the sinks of $C_1(i)$ are the sources of $\Xi_1(i)$ we have that in the time interval $[t_0, t_2]$ $c_3 m(i+1)$ sources of $\Xi_1(i)$ are pebbled, while at least $c_4 m(i)$ pebbles are constantly on the graph, thus proving the lemma for this case.

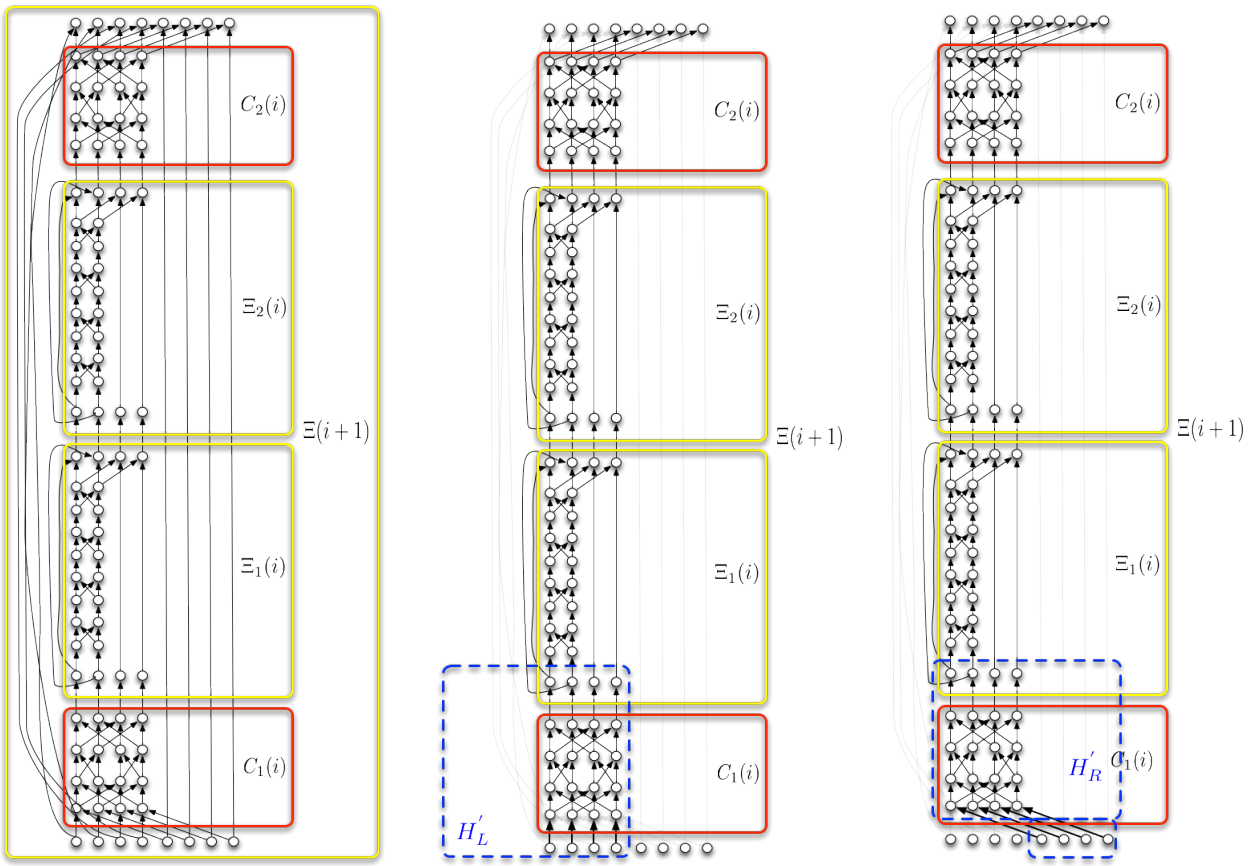


Table 3.2: The induced graphs used in this part of the proof

■

Special Case 2: Let $\mathcal{P}$ be a pebbling of $\Xi(i+1)$ meeting the lemmas' prerequisites and suppose that there exists a time interval $[t_1, t_2] \subseteq [0, t]$ such that at least $c_3 m(i)/2$ sources of the subgraph $\Xi_2(i)$ are pebbled and there are at least $c_2 m(i)$ pebbles on $\Xi(i+1)$ throughout the whole interval. Then the lemma's conclusions hold.

Proof. The proof is similar to the one in Case 1. We only need to adjust the induced graphs H_L' and H_R' in two new subgraphs H_L'' and H_R'' in the way shown in figure

3.3 (i.e. including the sinks of $\Xi_1(i)$, the sources of $\Xi_2(i)$, and take only the edges connecting directly the sources of $\Xi_1(i)$ directly to its' sinks, while bypassing the rest of $\Xi_1(i)$). Then we can apply the same reasoning as in the Special Case 1, 3.4.3.2. above.

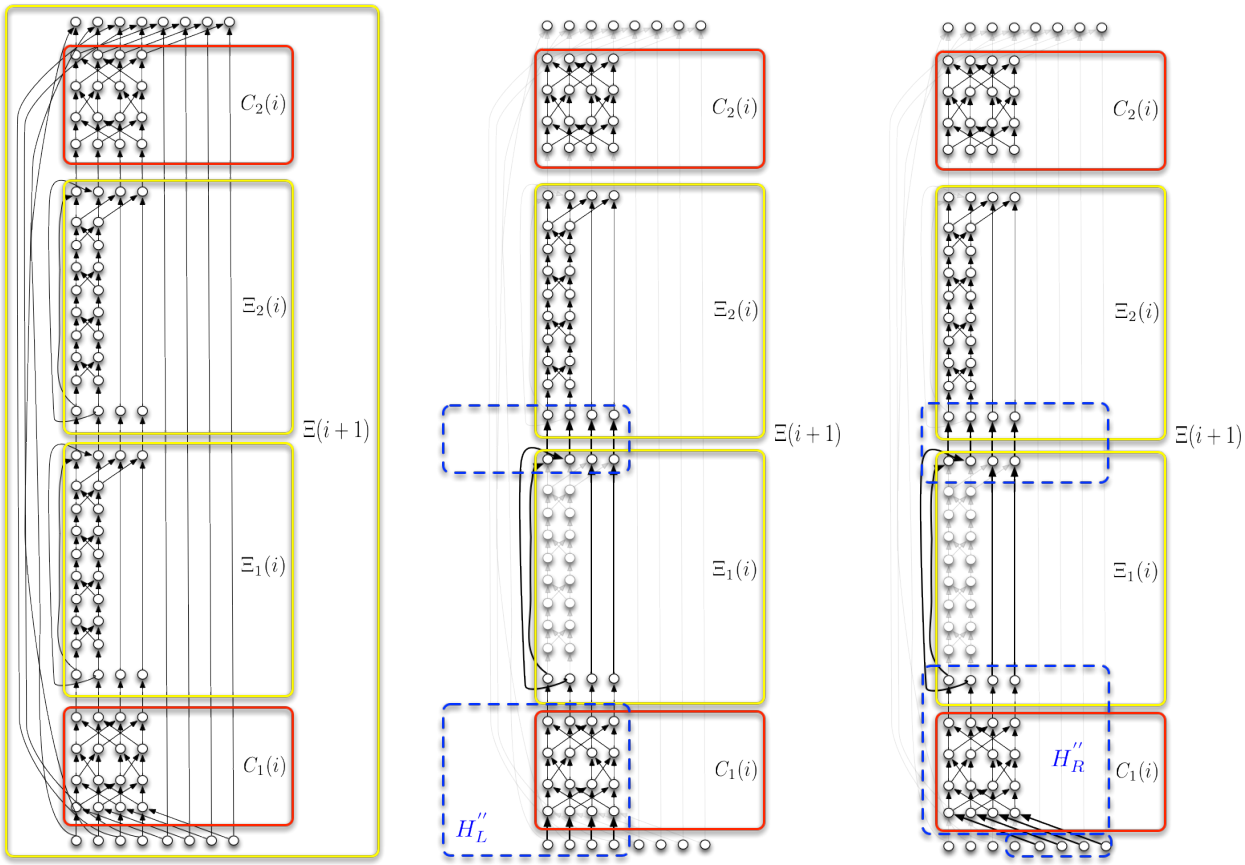


Table 3.3: The induced graphs used in this part of the proof

■

Special Case 3: Let $\mathcal{P}$ be a pebbling of $\Xi(i+1)$ meeting the lemmas' prerequisites and suppose that there exists a time interval $[t_1, t_2] \subseteq [0, t]$ such that at least $\frac{c_1}{2}m(i+1)$ sinks $\Xi(i+1)$ are pebbled while there are at least $c_2 m(i)$ pebbles on $\Xi(i+1)$. Then the lemma's conclusions hold.

Proof. We follow a similar reasoning as in the aforementioned special cases 1 and 2. We consider again two induced graphs H_L''' and H_R''', which are constructed as follows and can be seen in figure 3.4. H_L''' (and respectively H_R''') is the subgraph H_L''

constructed in 3.4.3.2 plus the sinks of $\Xi_2(i)$, the superconcentrator $C_2(i)$ and the left-hand half of the sinks of the PTC graph $\Xi(i+1)$. The edges of $H_L^{'''}$ are those of $H_L^{''}$ as well as the ones of $C_2(i)$, the edges connecting directly the sources of $\Xi_2(i)$ to its' sinks, bypassing all the edges of $\Xi_2(i)$, the edges connecting the sinks of $\Xi_2(i)$ to the sources of $C_2(i)$ and the edges connecting the sinks of $C_2(i)$ to the left-hand half sinks of $\Xi(i+1)$. As in the previous cases, we consider t_0 to be the last time before t_1, where there are no more than $c_2 m(i+1)$ pebbles on the graph. Then assuming that

$$\frac{c_1}{4} m(i+1) \geq c_2 m(i+1) + 1 \tag{3.4}$$

and since $H_L^{'''}$ and $H_R^{'''}$ are superconcentrators, the same reasoning as in 3.4.3.2 applies, thus proving the result.

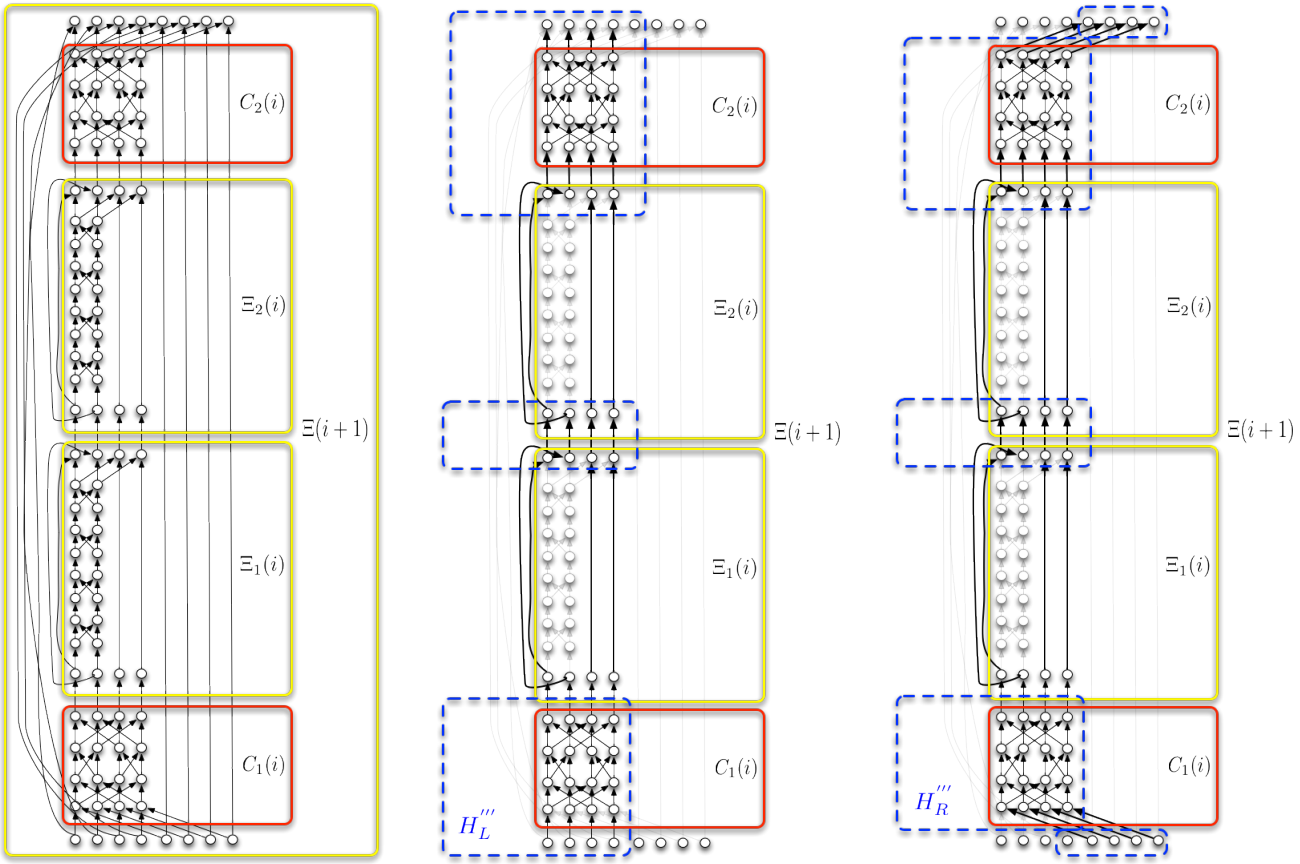


Table 3.4: The induced graphs used in this part of the proof

■

Note here that in the proof of 3.4.3.2 we could have as well used a specifically selected set of paths from the sources of the second superconcentrator $C_2(i)$ to its' sinks, thus avoiding this superconcentrators' inclusion to the subraphs H_L''' and H_R'''. So until now we have not yet seen the necessity of the second superconcentrator $C_2(i)$ to the construction of the PTC graphs. As we will move to the proof of lemma 3.4.3.4 using the induction hypothesis, the reason behind the need for the second superconcentrator will become clear.

We are now ready to proceed in proving lemma 3.4.3.4.

Proof. **Base Case:** Let i_0 be the first superconcentrator, which we will use for generating the PTC graph and consider an initial configuration of no more than $c_2m(i_0)$ pebbled vertices. Suppose now that $c_1m(i_0)$ sinks of the graph are pebbled during the time interval $[0,t]$. Then any $c_2m(i_0)+1$ of these sinks are connected to at least $m(i_0)-c_2m(i_0)$ sources via initially pebble free paths. Thus at least one of these sinks (let's call it v) is connected to at least $\frac{m(i_0)}{c_2m(i_0)+1}$ of these sources via initially pebble free paths. When v is pebbled, none of these $m(i_0)/(c_2m(i_0)+1$ sources is connected to v via a pebble-free path. Also the set of sources connected to v via a pebble-free path can decrease by at most one at each time step. Let t_1-1 be the last time at which $m(i_0)/(c_2m(i_0)+1$ sources are connected to v via pebble-free paths. During the time interval $[t_1,t]$,

$$\frac{m(i_0)}{c_2m(i_0)+1}-1$$

sources of $\Xi(i_0)$ must be pebbled, while at least one pebble is always on the graph. For the base case to hold we must demand that

$$\frac{m(i_0)-c_2m(i_0)-1}{c_2m(i_0)+1}\geq c_3m(i_0) \tag{3.5}$$

thus proving the lemma for the base step of the induction.

Induction Hypothesis: In $[0,t]$ at least $c_1m(i)$ sinks of $\Xi(i)$ are pebbled and at times 0 and t there are at most $c_2m(i)$ pebbles on the graph. Then there exists a time interval $[t_1,t_2]$ during which at least $c_3m(i)$ sources are pebbled and at least $c_4m(i)$ are on the graph.

Inductive step: Suppose that in $[0,t]$ at least $c_1m(i+1)$ sinks of $\Xi(i+1)$ are pebbled and at times 0 and t there are at most $c_2m(i+1)$ pebbles on $\Xi(i+1)$. We will show that there exists a time interval $[t_7,t_6]$ during which at least $c_3m(i+1)$ sources are pebbled and at least $c_4m(i+1)$ pebbles are always on the graph.

Suppose that none of the previous special cases holds. Then since case 3 (lemma 3.4.3.2) does not hold, there must be a time $t_1 \in [0,t]$ such that in $[0,t_1]$ fewer than $c_1m(i+1)/2$ sinks of $\Xi(i+1)$ are pebbled and $Space(\mathbb{P}_{\sigma_{t_1}}) \leq c_2m(i)$. Then this means that in $[t_1,t]$ at least $c_1m(i+1)/2$ sinks of $\Xi(i+1)$ are pebbled. Then by demand 3.4.3.2 we have that

$$\frac{c_1}{4}m(i+1) \geq c_2m(i+1)+1 \tag{3.6}$$

which clearly means that

$$c_1m(i) \geq c_2m(i)+1$$

and then since $C_2(i)$ is a superconcentrator, applying 3.4.3.3 we see that the number of sinks of $\Xi_2(i)$ connected to these $c_2m(i)+1$ sinks of $\Xi(i+1)$ via pebble free paths is at least $(1-c_2)m(i)$. Then demanding

$$(1-c_2m(i)) \geq c_1m(i) \tag{3.7}$$

we can apply the induction hypothesis on $\Xi_2(i)$ for the time interval $[t_1,t]$ (since $Space(\mathbb{P}_{\sigma_{t_1}}) \leq c_2m(i)$) and find a time interval $[t_2,t_3] \subseteq [t_1,t]$ during which $c_3m(i)$ sources of $\Xi_2(i)$ are pebbled and $c_4m(i)$ pebbles are always on the $\Xi_2(i)$.

Now in this time intervarl, there must exist a time t_4 such that fewer than $c_3/2m(i)$ sources of $\Xi_2(i)$ are pebbled in $[t_2,t_3]$ and the number of pebbles on $\Xi(i+1)$ at this time is less thatn $c_2m(i)$ because otherwise case 2 (lemma 3.4.3.2) would hold. Then at time t_4 there are at least $c_3/2m(i)-c_2m(i)$ sinks of $\Xi_1(i)$ connected to these sources of $\Xi_2(i)$ via pebble-free paths. Demanding now

$$\frac{c_3}{2}m(i) \geq c_1m(i) \tag{3.8}$$

we can apply the induction hypothesis to $\Xi_1(i)$ for the time interval $[t_4,t_3]$ and obtain a subinterval $[t_5,t_6]$ during which $c_3m(i)$ sources of $\Xi_1(i)$ are pebbled and at least $c_4m(i)$ pebbles are always on $\Xi_1(i)$.

Until now we have found a time interval $[t_5,t_6] \subseteq [0,t]$ in which $c_4m(i+1)$ pebbles are always on the graph $\Xi(i+1)$. What remains to be shown is that in this interval at least $c_3m(i+1)$ sources of $\Xi(i+1)$ are pebbled.

Since case 1 does not hold, then there exists a time $t_7 \in [t_5,t_6]$ such that in $[t_5,t_7]$ at most $c_3/2m(i)$ sources of $\Xi_1(i)$ are pebbled and the number of pebbles on $\Xi(i+1)$ at time t_7 ia less than $c_2m(i)$. Then in the interval $[t_7,t_6]$ at least $c_3/2m(i)$ sources of $\Xi_1(i)$ are pebbled. Now recall that $c_3/2m(i) \geq c_2m(i+1)$ (3.4.3.2) which means that $c_3m(i)/2 \geq c_2m(i)+1$. Notice then that in time t_7 at least $(1-c_2)m(i+1)$ sources

of $\Xi(i+1)$ are connected via pebble-free paths to the $c_3 m(i)/2$ sources of $\Xi_1(i)$. But then by 3.4.3.2 we have that

$$(1-2c_2)m(i+1) \geq c_3 m(i+1)$$

which means that in the time interval $[t_7, t_6]$ at least $c_3 m(i+1)$ sources of $\Xi(i+1)$ are pebbled, completing the proof.

■

Using the above lemma we can now prove that for the PTC superconcentrator family of graphs the following pebbling lower bound holds:

Theorem 3.4.3.2. *There exists a family of explicitly constructible DAGs* $\{G_n\}_{n=1}^{\infty}$ *with* $\Theta(n)$ *vertices, unique sink and indegree* 2*, such that* $BW - Peb(G) = \Omega(n/\log n)$.

Proof. Let $\Xi(i)$ be the PTC graph built on explicitly constructed superconcentrators and define the graph family $\{H_n\}_{n=1}^{\infty}$ by $H_n = \Xi(\lfloor \log n - \log\log n \rfloor)$. Then by lemma 3.4.3.4 we have that $BW - Pebb(H_n) = \Omega(n/\log n)$ and H_n has size $\Theta(n)$. For the single-sink version of this graph, it is easy to see that the same results apply. ■

3.4.4 Graph PoSE

The PTC graph using Butterfly graphs

The graph constructed in the previous section belongs to the CRR family of graphs, as long as the superconcentrator family used also does. Also since the number of vertices is $\Theta(i2^i)$ [2] (where $2^i = n$ is the number of sinks), running a BFS on the graph can pebble it in time $O(n \log n)$. Using lemma 3.4.3.4 makes also clear the fact, that any algorithm that pebbles it, must use $\Theta(n)$ space. It is therefore reasonable to assume that this family of graphs has all the desired properties for constructing an efficient PoSE:

- It is compactly recursively representable
- It requires $\Theta(n)$ space
- It can be pebbled in $O(n \log n)$ time

Unfortunately a closer look to the proof of lemma 3.4.3.4 reveals that the situation is far from ideal:

[2] Solving the recursive relationship

$$V(i) = 2(2^i + 2^{i-1}2(i-1) + V(i-1)$$

, where $V(i)$ is the number of vertices of the PTC graph with 2^i sinks and $2^{i-1}2(i-1)$ is the number of vertices added by the $But(1^{i-1}$, proves the claim

Lemma 3.4.4.1. *Any algorithm that pebbles a [PTC76] superconcentrator of* 2^i *sinks/sources uses at least* 2^{i-6} *pebbles.*

Before proving the above lemma, we show the next lemma which will play an important role in the formers' proof:

Lemma 3.4.4.2. *Any pebbling strategy that pebbles at least 14 sinks of the butterfly graph with 64 sinks/sources (*$But(64)$*) and starts with a configuration of 3 pebbles, pebbles at least 34 sources, while maintaining at least one pebble throughout the whole time.*

Proof. We will prove this lemma by actually proving something stronger: namely we argue that in order to pebble 1 sink, all the sources of the graph need to be succintly pebbled. We start by observing that in order to pebble a single sink of the $But(4)$, any strategy would have to pebble all the 4 sources of the graph. Then since any sink of $But(8)$ needs to pebble two sinks of the two distinct $But(4)$ subgraphs and all of the latters' sources need to be pebbled, then for 1 sink of $But(8)$ to be pebbled, all the 8 sources need to be pebbled as well. The idea propagates in the next members of the butterfly graph family, thus proving that for one sink of the $But(64)$ graph to be pebbled, all its' 64 sources need to be pebbled. Then clearly the lemma holds for the case of this graph.

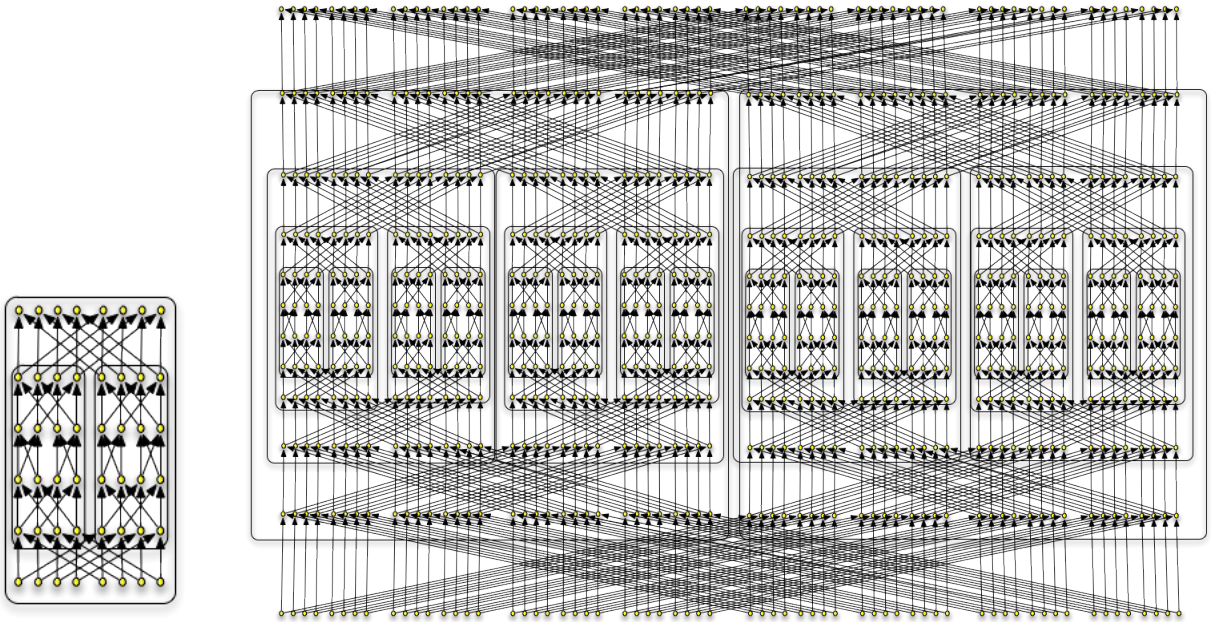


Table 3.5: The $But(8)$ and $But(64)$ graphs

■

Proof. In the proof of lemma 3.4.3.4 we had to demand certain inequalities in order to be able to apply certain superconcentrators facts. These are repeated here, along with a brief reminder of their necessity, for ease of reading:

$$\frac{m(i_0) - c_2 m(i_0) - 1}{c_2 m(i_0) + 1} \geq c_3 m(i_0) \tag{3.9}$$

$$\frac{c_3}{2} m(i) \geq c_2 m(i+1) + 1 \tag{3.10}$$

$$N(0) - 2c_2 \geq c_3 \tag{3.11}$$

$$c_2 m(i) - 1 \geq c_4 m(i+1) \tag{3.12}$$

$$\frac{c_1}{4} m(i+1) \geq c_2 m(i+1) + 1 \tag{3.13}$$

$$(1 - c_2) m(i) \geq c_1 m(i) \tag{3.14}$$

$$\frac{c_3}{2} m(i) \geq c_1 m(i) \tag{3.15}$$

$$\tag{3.16}$$

We had to assume that **??** holds so that the base case of the induction would hold. We assumed 3.10 and 3.13 in order to apply 3.4.3.3 to $C_1(i)$ and $C_2(i)$ respectively. We assumed 3.11 and 3.12 for the results to hold in the case 1 (3.4.3.2). We assumed 3.14 in order to apply the induction hypothesis on $\Xi_2(i)$ and 3.15 in order to apply the induction hypothesis in $\Xi_1(i)$ and conclude the number of sources of $\Xi(i+1)$ pebbled in the time interval we had found. In [PTC76] these constants are set to $m(i_0) = 2^8$, $c_1 = 14/256$, $c_2 = 3/256$, $c_3 = 34/256$ and $c_4 = 1/256$. Regarding c_2 we cannot do much, since the in- and out- degree of the graph is bounded by 2. Ideally we would like to lower the number of sinks/sources of the recursions' base case, $m(i_0)$. And indeed we can (however not dramatically) ask for $m(i_0) = 64$, if instead of using any other superconcetrator family, we use the butterfly graphs illustrated in 3.4.4. Then we can apply 3.4.4.2 and then we can check constants $c_1 = 14/64$, $c_2 = 3/64$, $c_3 = 34/64$ and $c_4 = 1/64$ satisfy the conditions stated earlier. ∎

Let G be a single-sink/single source $N-$superconcentrator. We construct the following Proof of Secure Erasure, based on the ideas described in the previous sections:

1. $\mathcal{V}$ sends to $\mathcal{P}$ G's description and the label of its source S_0 and measures the time that $\mathcal{P}$ needs in order to pebble G and output the label of G's sink vertex.

2. $\mathcal{P}$ pebbles G and sends back to $\mathcal{V}$ the label of G's sink.

3. $\mathcal{V}$ accepts if and only if $\mathcal{P}$ calculated the sinks' label correctly within the expected timeframe.

Instead of the butterfly graphs, we could of course use any other superconcentrator family with linear vertices in the number of the sources. As a previous step in our graphPoSE construction we prove in the Appendix, that the superconcentrator family proposed in [AC03] is CRR and therefore could be used in building the PTC graphs. The construction of this superconcentrator family is inductive but the base begins with a number of sources equal to 2^{18} which may not be suitable for a great variety of the cases that we will be willing to apply our protocol. Therefore for our graphPoSE we use the family of the butterfly graphs, for which the lemmata in this section hold.

Security Proof

In this section we describe the algorithm that is ran by an honest prover and prove the graphPoSE to be secure under the definition 3.1.2.

Assume that each node's encoding is of the form $(index, Label, layer)$ and that for the labeling of each node a hash function $H : \{0,1\}^{2w} \to \{0,1\}^{w}$ is used. Then using the following algorithms we can pebble the i-th layer of a given GT butterfly graph (which has 2^i sinks/sources) in space $2^{i-1} + 3$ and time $i2^i$, given each node's in- and out-degree equaling 2 as well as the butterfly graph's structure.

Algorithm 5 Pebbler(S_0)

Input: The unique sinks' label, S_0
for $j = 1$ to $j = 2^i$ **do**
  $L[j] \leftarrow (j, label(j), 0)$
end for
for $j = 1$ to $j = 2^i$ **do**
  $tmp1 \leftarrow label(2^{i+j})$
  $tmp2 \leftarrow L[j].label()$
  $L[j] \leftarrow (j, tmp2, 1)$
end for
for $j = 1$ to $j = \log n$ **do**
  for all $a_i \in L$ **do**
    if $a_i.layer == k$ **then**
      $tmp1 \leftarrow a_1.Child1$\{only index needed\}
      $tmp2 \leftarrow a_1.Child2$\{only index needed\}
      $tmp3 \leftarrow tmp1.Parent2$\{assume that $Parent1$ is a_i and $tmp3$ is some a_h\}
      $tmp4 \leftarrow H(a_1.Label(), a_h.Label())$
      $a_1 \leftarrow (tmp1, tmp4)$
      $a_1.layer++$
      $tmp4 \leftarrow H(a_h.Label(), a_1.Label())$
      $a_h \leftarrow (tmp2, tmp4)$
      $a_h.layer++$
    else
      Go to the next element in L
    end if
  end for
end for

Algorithm 6 i-Pebbler

Input: A GT graph with 2^i sources
Storage capacity: 2^{i+1}
Output: The label of the unique sink
Pebble the sources of $C_2(i-1)$
Use the additional 2^i registers to hold the labels of the initial sources

Lemma 3.4.4.3. *The graphPoSE is* 1/32*-robust*

Proof. The lemma holds using lemmata 3.4.4.1 as well as algorithm 5 ∎

Chapter 4

Conclusions & Future Work

In the following table, we show the results and compare the PoSEs that have been proposed in the literature, along with our two constructions. One can see that the problem of erasing securily all the contents of a devices' memory is still open: On one hand Tsudik's solution guarantees the erasure in the standard model, but pays in communicating complexity. Diembowskis' solution also guarantees secure erasure, but the time complexity is very high, while at the same time the protocol is being proven secure in the random oracle model. Our iHash guarantees secure erasure but only in a specific adversarial model. And the graphPoSE guarantees erasure of only a fracture of the provers' memory. This is only the beginning of the research in this field, which due to its practical applications seems like it will receive more interest in the near future. When this done, these protocols could become part of any code update.

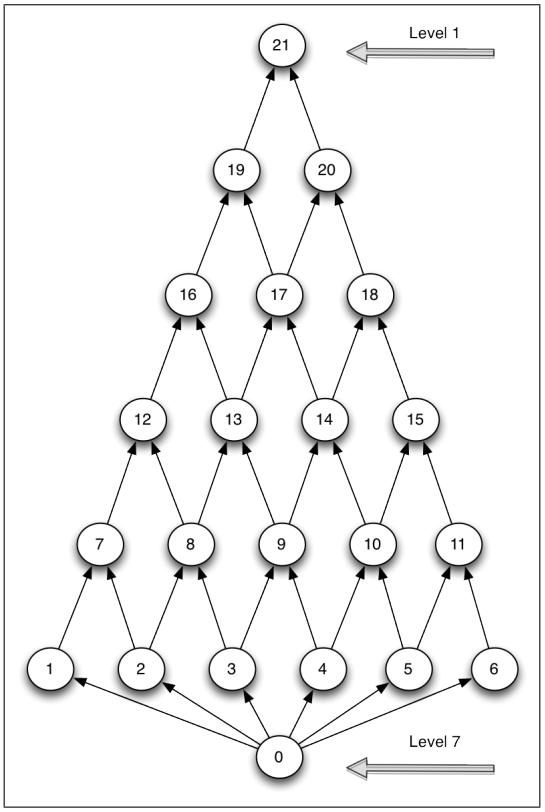
	Communication Complexity	Time	α
[PT10]	S	S	$1-\epsilon$
[DKW11]	$\log \gamma^{-1}$	S^2	$1-\epsilon$
iHash	$1.39(1-\alpha)^{-1}\log\gamma^{-1}$	$S\log S \cdot 1.39(1-\alpha)^{-1}\log\gamma^{-1}$	any
graphPoSE	$\log \gamma^{-1}$	$S\log S$	1/32

Appendix A

Arrowhead Functions are CRR

Let G be an arrowhead graph, as described in [DKW11] and n be the maximum number of nodes, that we can have in memory. Then the total number of nodes for an arrowhead graph is $n(n+1)/2$ (excluding the input node). In order to run the protocol described in [DKW11], the prover has to create G. However in order to represent this graph, the prover has to use more memory than what he has available. In this section we give a polynomial time and constant space algorithm, that given a node of an arrowhead function, returns its incoming edges, thus proving that the arrowhead head functions used in [DKW11] are compactly recursively representable.

We name the nodes in increasing order starting from 0 for the input node and ending with $n(n+1)/2$ for the output node. An instance of a resulting graph can be seen in the following picture:



Let level 1 be the level of the output node, increasing to level $n+1$, which will be the input nodes' level. Then we can see that for a node u, its' right parent is $u - level$ and its' left parent is $u - level - 1$, unless the nodes' level is equal to n, in which case we set its' left parent equal to NULL and its' right parent equal to 0. So all one needs to do is to find the nodes' level, which in order to do, it suffices to run the following algorithm:

Algorithm 7 LevelFinder(u, n)

```
1: Input: u the node's name whose level we want to find,
   n the input node's number of children
2: A = n
3: if A ≥ u then
4:    level = n
5: else
6:    for i = 1 to n do
7:       A = A + n − i
8:       if A ≥ u then
9:          level = n − i
10:         break
11:      end if
12:   end for
13: end if
14: return level
```

The above algorithm clearly runs in polynomial time and in constant space, thus proving the statement.

Appendix B

Graph PoSE with other Superconcentrators

We begin by describing the $N-$superconcentrator construction given in [AC03]: First we need to construct a 9-regular Ramanujan graph with $k \cdot 2^{18l-c}$ vertices. To do this we need to:

1. Select an irreducible polynomial of degree 2, g in the field of 8 elements, $\mathbb{F}_8[x]$ and construct the field $K_l = \mathbb{F}_8[x]/g\mathbb{F}_8[x]$

2. Construct the group $\mathrm{H}_l = \mathrm{PGL}_2(K_l)$ [1].

3. Fix a γ in $\mathbb{F}_8[x]$ such that the resulting polynomial $q(x) = x^2 + x + \gamma$ is irreducible in $\mathbb{F}_8[x]$ and let β_l be a root of $q(x)$ in $\mathbb{F}_8[x]/g^l(x)F_8[x]$. Let Σ_l be the following subset of H_l:

$$\Sigma_l = \left\{ \left(\begin{array}{cc} 1 & \epsilon + \delta\beta_l \\ (\epsilon + \delta\beta_l + \delta)x & 1 \end{array} \right) : \delta, \epsilon \in \mathbb{F}_8, \epsilon^2 + \epsilon\delta + \delta^2\gamma = 1 \right\} \tag{B.1}$$

   Then Σ_l has 9 elements, each of degree 2 in $\mathrm{PGL}_2(K_l)$

4. Consider the set

$$\mathrm{H}' = \left\{ \left(\begin{array}{cc} 1 + g^l(x)s & g^l(x)r \\ tg^l(x) & 1 \end{array} \right) : r, s, t \in \mathbb{F}_8[x]/g\mathbb{F}_8[x] \right\}$$

[1] recall that $\mathrm{PGL}_2(K_l) = \mathrm{GL}_2(K_l)/\mathcal{Z}(K_l)$, where $\mathcal{Z}(\mathrm{GL}_2(\mathrm{PGL}_2(K_l))) = \{cI, c \in K_l\}$ and I the 2×2 identity matrix

which forms a group under multiplication and is isomorphic to $\mathbb{F}_2^{18}$ [2]. Then for every $c \leq 18$ there exists a subgroup $\mathrm{H}'_c \leq \mathrm{H}$.

5. The Schreier graph $\Lambda_l = (\mathrm{H}_{l+1}; \mathrm{H}_{l+1}/\mathrm{H}'_c; \Sigma_{l+1})$ is a 9-regular Ramanujan graph with $k \cdot 2^{18l}$ vertices.

Lemma B.0.4.4. *The family of explicitly constructed 9-regular Ramanujan graphs presented in [AC03] is Compactly Recursively Representable.*

Proof. First we show that the 9-regular Ramanujan graph Λ_l is CRR: Let Λ_l be the Schreier graph $(\mathrm{H}_{l+1}; \mathrm{H}_{l+1}/\mathrm{H}'_c; \Sigma_{l+1})$. For ease of reading, for a polynomial $f(x)$ we will write simply f. The vertices of this graph will be cosets, whose elements will be 2×2 matrices (represented as 4 dimensional vectors) of the form

$$\begin{pmatrix} a_1 + gp_1 + (a_1 s + a_2 t)g^l & a_2 + gp_2 + ra_1 g^l \\ \\ a_3 + gp_3 + (a_3 s + a_4 t)g^l & a_4 + gp_4 + ra_3 g^l \end{pmatrix} \tag{B.2}$$

where $a_j, s, r, t \in K_1 = \mathbb{F}_8[x]/g(x)\mathbb{F}_8[x]$ and $p_j \in K_l$, $j \in \{1,2,3,4\}$ and the edges will be of the form $(u, u \cdot \sigma_i)$, where u is a coset $g_1\mathrm{H}_{l+1}/\mathrm{H}'_c$, $g_1 \in \mathrm{H}_{l+1}$, $\sigma_i \in \Sigma_l$ and $i \in \{1, \cdots, 9\}$. We will show that for every $c \leq 18$, there exists a family of vertices, called coset leaders, with the following properties:

1. Every coset has exactly one member in this family.

2. For every coset it is easy to find its leader, given a member of the coset.

We examine the case where $c = 18$. Observe that an element of the coset will be of the form

$$\begin{pmatrix} \upsilon_1 + (a_1 s + a_2 t + \pi_1)g^l & \upsilon_2 + (ra_1 + \pi_2)g^l \\ \upsilon_3 + (a_3 s + a_4 t + \pi_3)g^l & \upsilon_4 + (ra_3 + \pi_4)g^l \end{pmatrix}, \tag{B.3}$$

where $a_j, s, r, t \in K_1$, $p_j \in K_l$ and υ_j, π_j are the remainder and the quotient respectively, of $a_j + gp_j$ divided by g^l, $j \in \{1,2,3,4\}$. By the construction of K_1 we have that (since $a_1 a_4 \neq a_2 a_3$) it is always possible to find s and t such that:

$$\begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix} \begin{pmatrix} s \\ t \end{pmatrix} = \begin{pmatrix} -\pi_1 \\ -\pi_3 \end{pmatrix} \tag{B.4}$$

[2] H' is the kernel of a surjective homomorphism $\phi : \mathrm{H}_{l+1} \to \mathrm{H}_l$ such that

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \mapsto \begin{pmatrix} a'_{11} & a'_{12} \\ a'_{21} & a'_{22} \end{pmatrix}$$

where $a'_{ij} = a_{ij} \mod g^l(x)$ for each $i, j \in \{1,2\}$.

If $a_1 \neq 0$ then by setting $r = -\pi_2/a_1$, the coset leader will be:

$$\begin{pmatrix} \upsilon_1 & \upsilon_2 \\ \upsilon_3 & \upsilon_4 + \left(\pi_4 - \frac{\pi_2 a_3}{a_1}\right) g^l(x) \end{pmatrix}, \tag{B.5}$$

and if $a_1 = 0$ we have that $a_3 \neq 0$ and setting $r = -\pi_4/a_3$ we set the coset leader to be:

$$\begin{pmatrix} \upsilon_1 & \upsilon_2 + \pi_2 g^l(x) \\ \upsilon_3 & \upsilon_4 + (\pi_4 + r a_3) g^l \end{pmatrix}, \tag{B.6}$$

Identifying a coset leader is easy, since by its form in (B.5) we see that $\upsilon_1, \upsilon_2, \upsilon_3$ will be encoded as bitstrings, whose bits after position $2l$ will be zeros, while υ_4 will have at least one bit equal to 1 after the position $2l$. Therefore in order to find out if an element is the coset leader or not, we just need to check that in $\upsilon_1, \upsilon_2, \upsilon_3$ all bits after the position $2l$ are zeros and there is at least one bit different than zero, after the $2l$ position in υ_4.
Next we show that the coset leader is unique for every coset: Let $g_1 \mathrm{H}'_{18}$ be a coset and suppose that there exist two different coset leaders,

$$\begin{pmatrix} \upsilon_1 & \upsilon_2 \\ \upsilon_3 & \upsilon_4 + \left(\pi_4 - \frac{\pi_2 a_3}{a_1}\right) g^l(x) \end{pmatrix} \text{ and } \begin{pmatrix} \upsilon_1^* & \upsilon_2^* \\ \upsilon_3^* & \upsilon_4^* + \left(\pi_4^* - \frac{\pi_2^* a_3^*}{a_1^*}\right) g^l(x) \end{pmatrix}$$

Then it is easy to see that $\upsilon_i = \upsilon_i^*$, and $\pi_i = \pi_i^*$ for $i \in \{1, 2, 3, 4\}$ because of the uniqueness of polynomial division.
The last step that we need to show, is that if l_1 and l_2 are coset leaders from different cosets, found from two elements x and y respectively and $l_1 = l_2$, then $x = y$.

Let $x \in g_1 \mathrm{H}'_{18}$, $y \in g_2 \mathrm{H}'_{18}$,

$$g_1 = \begin{pmatrix} a_1 + gp_1 & a_2 + gp_2 \\ a_3 + gp_3 & a_4 + gp_4 \end{pmatrix}, \; g_2 = \begin{pmatrix} a_1^* + gp_1^* & a_2^* + gp_2^* \\ a_3^* + gp_3^* & a_4^* + gp_4^* \end{pmatrix}$$

and

$$x = \begin{pmatrix} x_1 & x_2 \\ x_3 & x_4 \end{pmatrix} = \begin{pmatrix} \upsilon_1 + (a_1 s + a_2 t + \pi_1) g^l & \upsilon_2 + (r a_1 + \pi_2) g^l \\ \upsilon_3 + (a_3 s + a_4 t + \pi_3) g^l & \upsilon_4 + (r a_3 + \pi_4) g^l \end{pmatrix}$$

$$y = \begin{pmatrix} y_1 & y_2 \\ y_3 & y_4 \end{pmatrix} = \begin{pmatrix} \upsilon_1^* + (a_1^* s + a_2^* t + \pi_1^*) g^l & \upsilon_2^* + (r a_1^* + \pi_2^*) g^l \\ \upsilon_3^* + (a_3^* s + a_4^* t + \pi_3^*) g^l & \upsilon_4^* + (r a_3^* + \pi_4^*) g^l \end{pmatrix}$$

By our assumption we have that $l_1 = l_2$, or in other words:

$$\begin{pmatrix} v_1 & v_2 \\ v_3 & v_4 + (\pi_4 - \pi_3 a_3/a_1)g^l \end{pmatrix} = \begin{pmatrix} v_1^* & v_2^* \\ v_3^* & v_4^* + (\pi_4^* - \pi_3^* a_3^*/a_1^*)g^l \end{pmatrix}$$

which means that $v_i = v_i^*$ for $i \in \{1, 2, 3, 4\}$,

Equivalent arguments hold for the case where $a_1 = 0$ and the coset leader is of the form (B.6).

Given the above, the algorithm for finding the coset leader, given an element in the coset (CosetLeaderFinder(u), where u is the given coset element) is straightforward:

Algorithm 8 CosetLeaderFinder(v)

1: Input: A coset member of the form (B.3)
2: Solve the system in (B.4)
3: **if** $a_1 \neq 0$ **then**
4: **return** the coset leader of the form (B.5)
5: **else**
6: **return** the coset leader of the form (B.6)
7: **end if**

and the following algorithm for finding the predecessors of a given vertex $v \in V$ can be implemented by a circuit of polylogarithmic size and is as follows:

Algorithm 9 SuperConcentratorPredecessorFinder(v)

1: Initialize a queue Q
2: $u =$ CosetLeaderFinder(v)
3: **for** $\sigma_i \in \Sigma_l$ **do**
4: $t = u \cdot \sigma_i$
5: $t =$ CosetLeaderFinder(t)
6: $Q \leftarrow t$
7: **end for**

■

Definition B.0.4.1. Let x be a coset leader of the form (B.5) or (B.6). Define its representational number (reprNumber) be the integer which has the binary representation of the concatenation of $v_1, v_2, v_3, v_4+(ra_3+\pi_4)g^l$ in the first case or the binary representation of the concatenation of $v_2 + \pi_2 g^l(x), v_3, v_4$ in the latter case.

Lemma B.0.4.5. *The family of explicitly constructed Superconentrators presented in [AC03] is Compactly Recursively Representable.*

Proof. Let X be a set of coset leaders defined in (B.0.4.4) and $|X| = N = 262,080 \cdot 2^l$. The graphs' vertices will consist of four copies of X, X_1, X_2, X_3, X_4. In the representation of every coset leader we will add an index, that signifies the copy of X, in which this coset leader belongs. The vertices in X_1 will be the input nodes of the superconcentrator and the vertices of X_4 will be the output nodes of the superconcentrator. The edges will be added in the following manner (where an edge (a, b) will be directed from a to b):

- For a vertex $u \in X_4$ its incomming edges are the set:

$$\{(u \cdot \sigma_i, u, 3), \sigma_i \in \Sigma_l \text{ as it is defined in (B.1)}\} \cup (u, u, 3)$$

- For a vertex $u \in X_3$ we have the following two cases:
  - If numRepr$(u) \leq N/2$ then its incomming edges will be $(v, u, 3)$, $(v, u, 2)$, where numRepr(v) = numRepr$(u)+N/2$ and the 9 incoming edges when considering the Ramanujan graph with output nodes, the ones in X_3 that have numRepr $\leq N/2$ and input nodes the ones in X_2 that have numRepr $\leq N/2$.
  - If numRepr$(u) > N/2$ then its only incoming edge will be $(v, u, 2)$, where numRepr(v) = numRepr$(u) - N/2$

  For a vertex $u \in X_2$ we have again two cases:
  - If numRepr$(u) \leq N/2$, then its incomming edges will be the set

$$\{(u \cdot \sigma_i, u, 1), \sigma_i \in \Sigma_l \text{ as it is defined in (B.1)}\} \cup (u, u, 1)$$

    and $(v, u, 2)$, where numRepr(v) = numRepr$(u) + N/2$.
  - If numRepr$(u) > N/2$, then its incomming edges will be the set $\{(u \cdot \sigma_i, u, 1), \sigma_i \in \Sigma_l$ as it is defined in (B.1)$\} \cup (u, u, 1)$.

■

Remark. For the graphs to work in the setting we want, the superconcentrator must have bounded indegree 2. However in the above construction, the nodes' indegree vary from 9 to 11. This can be easily fixed by making the following adjustment, which we present for the case of indegree 9 and can be easily extended to the cases of 10 and 11.

In the description of a vertex v we will add the 9 elements $\sigma_i \in \Sigma_l$ defined in (B.1) along with a pointer, which we will symbolize here with $*$. v will be connected with an incoming edge to a vertex whose description will be $(v \cdot \sigma_9, \sigma_1, \cdots, \sigma_9)$ and to a vertex u_7^v whose description will be $(v, \sigma_1, \cdots, \sigma_8, \sigma_9^*)$. u_7^v will be connected to u_6^v, whose description will be

$(v, \sigma_1, \cdots, \sigma_8^*, \sigma_9)$. In a similar manner we will make the connections to $u_6^v, \cdots, u_2^v$ and we will connect u_1^v to the two vertices with description $(v \cdot \sigma_1, \sigma_1, \cdots, \sigma_9)$, $(v \cdot \sigma_2, \sigma_1, \cdots, \sigma_9)$. A graphical depiction of the above described procedure follows.

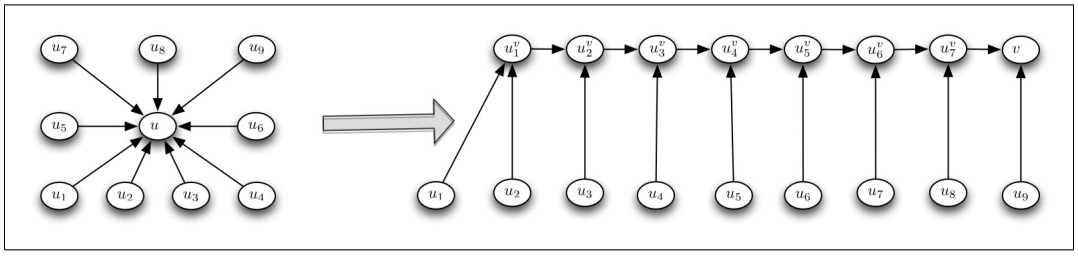


Figure B.1: Changing bounded indegree 9, to indegree 2

In a similar manner we can connect all the output nodes to one single output node, preserving the constant indegree 2 property for every graph node. We therefore have the following

Definition B.0.4.2. (Single-sink/Single-source Superconcentrator). Let G be an $N-$superconcentrator with sinks $Z(G) = \{z_1, \ldots, z_N\}$ for $N > 1$ and sources $S(G) = \{s_1, \ldots, s_N\}$. The single-sink version G, consists of all vertices and edges in G plus the extra vertices $\{z_1^*, \ldots, z_N^*\}$ and the edges $\{(z_1, z_1^*), (z_2, z_1^*), (z_1^*, z_2^*), (z_3, z_2^*), (z_2^*, z_3^*), (z4, z3^*\})$, etc. up to $\{(z_{N-2}^*, z_{N-1}^*), (z_N, z_{N-1})\}$ and an extra vertex S_0 along with the edges $\{(S_0, s_1), (S_0, s_2), \ldots, (S_0, s_N)\}$.

Lemma B.0.4.6. *The Gilbert-Tarjan graphs using the superconcentrators from [AC03] are CRR.*

Proof. Let G be Gilbert-Tarjan graph with $N(l) = K \cdot 2^{l+1}$ output nodes. We divide G in 6 layers, namely:

- layer 1, which contains the graphs' input nodes,
- layer 2 which is the first copy of the superconcentrator C_i^1,
- layer 3 which is the first copy of the Gilbert-Tarjan graph with $N(l)$ nodes, Ξ_i^1,
- layer 4 which is the second copy of the Gilbert-Tarjan graph with $N(l)$ nodes, Ξ_i^2,
- layer 5 which is the first copy of the superconcentrator C_i^2 and
- layer 6 which contains the graphs' output nodes.

In every nodes' description we will add an index $j, j \in \{1, 2, \cdots, 6\}$ which will indicate the layer at which the node resides. Let $P_i(x)$ be the i-th ($i \in \{1, 2\}$) predecessor of node x. Then the algorithm recursive algorithm, which returns x's predecessors is:

Algorithm 10 PredFinder(x)

if *layer* $\neq 2$ **then**
  if $x > N(l)$ **then**
    $P_1(x) = (x - N(l), 5)$
    $P_2(x) = (x, 1)$
  else
    $P_1(x) = (x, 5)$
    $P_2(x) = (x, 1)$
  end if
else
  $P_1(x) = (x, 1)$
  $P_2(x) = (x + N(l), 1$
end if

Combining (B.0.4.5), (B.0.4.1) and (B.0.4.2) we attain the result.

By the above construction we have that the representation of a node of the graph will be a vector of the form:

$$((a_i)_{i=1}^4, (\sigma_j)_{j=1}^9, k, l, m)$$

where $a_i, i \in \{1, 2, 3, 4\}$ is the coset leader representation (B.5) or (B.6), $\sigma_j, j \in \{1, \ldots, 9\}$ are the elements in Σ_l as described in (B.0.4.1) and (B.1), k will be the representational number of $(a_i)_{i=1}^4$ as defined in (B.0.4.1), l will be the layer of the superconcentrator and m will be the layer of the Gilbert-Tarjan graph. ■

Bibliography

[AC03] Noga Alon and Michael R. Capalbo. Smaller explicit superconcentrators. pages 340–346, 2003.

[DKW11] Stefan Dziembowski, Tomasz Kazana, and Daniel Wichs. One-time computable self-erasing functions. 6597:125–143, 2011.

[FN99] Amos Fiat and Moni Naor. Rigorous time/space trade-offs for inverting functions. *SIAM J. Comput.*, 29(3):790–803, 1999.

[Hel80] Martin E. Hellman. A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*, 26(4):401–406, 1980.

[HS74] Ellis Horowitz and Sartaj Sahni. Computing partitions with applications to the knapsack problem. *J. ACM*, 21(2):277–292, 1974.

[Nor11] Jakob Nordström. New wine into old wineskins: A survey of some pebbling classics with supplemental results. 2011.

[Pip77] Nicholas Pippenger. Superconcentrators. *SIAM J. Comput.*, 6(2):298–304, 1977.

[PT10] Daniele Perito and Gene Tsudik. Secure code update for embedded devices via proofs of secure erasure. 6345:643–662, 2010.

[PTC76] Wolfgang J. Paul, Robert Endre Tarjan, and James R. Celoni. Space bounds for a game of graphs. In Ashok K. Chandra, Detlef Wotschke, Emily P. Friedman, and Michael A. Harrison, editors, *STOC*, pages 149–160. ACM, 1976.

[Rab10] Tal Rabin, editor. *Time Space Tradeoffs for Attacks against One-Way Functions and PRGs*, volume 6223 of *Lecture Notes in Computer Science*. Springer, 2010.

[Sch06] Uwe Schöning. Smaller superconcentrators of density 28. *Inf. Process. Lett.*, 98(4):127–129, 2006.

[SS81] Richard Schroeppel and Adi Shamir. A t=o($2^{n/2}$), s=o($2^{n/4}$) algorithm for certain np-complete problems. *SIAM J. Comput.*, 10(3):456–464, 1981.

[Wag02] David Wagner. A generalized birthday problem. 2442:288–303, 2002.

[Yeh08] Katz Jonathan & Lindell Yehuda. *Introduction to modern Cryptography.* Chapman & HallCRC, 2008.