Towards bridging theory and implementation of cryptographic primitives

A dissertation presented by

Eric Miles

to the Faculty of the Graduate School
of the College of Computer and Information Science
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Northeastern University
Boston, Massachusetts
April, 2014
This dissertation is based on research supported by the National Science Foundation under grants CCF-0845003 and CCF-1319206.

Towards bridging theory and implementation of cryptographic primitives
© 2014 – Eric Miles
NORTHEASTERN UNIVERSITY
GRADUATE SCHOOL OF COMPUTER SCIENCE
PH.D. THESIS APPROVAL FORM

THESIS TITLE: Towards bridging theory & implementation of cryptographic primitives

AUTHOR: Eric Miles

Ph.D. Thesis Approved to complete all degree requirements for the Ph.D. Degree in Computer Science.

[Signatures and dates]

Distribution: Once completed, this form should be scanned and attached to the front of the electronic dissertation document (page 1). An electronic version of the document can then be uploaded to the Northeastern University-UMI website.
Abstract

It has been widely observed that there is a significant gap between the way that many cryptographic primitives are constructed and analyzed by the theory/foundations community, and the way that they are implemented, used, and attacked in practice. In this dissertation we study from two different perspectives the complexity of constructing cryptographic primitives, with an eye towards bridging this gap.

We first study the complexity of constructing pseudorandom functions using a paradigm known as the substitution-permutation network (SPN). Informally, a pseudorandom function family (PRF) is a small set of functions $F$ such that a function chosen at random from $F$ is indistinguishable from a truly random function by a computationally-bounded adversary with oracle access to the function. The SPN paradigm is widely used in practical cryptographic constructions, such as the Advanced Encryption Standard [Daemen & Rijmen 2000], but has not previously been used to construct candidate PRFs. We construct several new candidate PRFs inspired by the SPN paradigm, and show that they are computable more efficiently than previous candidates in a variety of computational models.

We then study the complexity of constructing arbitrary cryptographic primitives in an attack model in which the adversary obtains more information than just the input/output behavior afforded by oracle access. This line of research is motivated by the many practical side-channel attacks that exploit implementation properties, i.e. properties of the hardware on which an algorithm is run, rather than the algorithm alone. As a general result, we show how to efficiently compile any circuit $C$ into an equivalent leakage-resilient circuit $\hat{C}$ such that any function on the wires of $\hat{C}$ that leaks information during a computation $\hat{C}(x)$ gives a related function that compresses iterated group products over the alternating group $A_n$. In combination with new lower bounds for this problem which we prove and which may be of independent interest, our compiler withstands leakage from $\text{NC}^1$ assuming $\text{NC}^1 \neq \text{L}$, and unconditionally withstands leakage from virtually any class of functions against which average-case lower bounds are known.
Acknowledgements

This thesis is dedicated to the many people without whose love, support, and friendship it would not exist.

To my advisor, Emanuele Viola, who taught me to be a researcher. I benefited enormously from his brilliance and his boundless time and attention; he was patient with me when I needed it, impatient when I needed that too, and I will forever be grateful for his mentorship.

To my parents, Dana, Nancy, and Dave, who have always guided me and given me a context in which to understand the world. I cannot imagine a more inspiring set of role models, both in my professional life and my personal relationships.

To my friends at Northeastern, who made the difficult times bearable and the good times worthwhile. Any list here would be incomplete, but I am especially thankful to Dan Brown, Rahman Doost, Ancsa Hannak, Zahra Jafargholi, Hamid Jahanjou, Arash Molavi, Scott Roche, Tim Smith, Vincent St-Amour, Zhifeng Sun, and most of all Dimitris Kanoulas, who was with me at the beginning, the end, and each day in between.

To my friends from Berkeley, who continue to enrich my life with their humor and insight. I would be remiss without specifically mentioning Alex Jaffe, who made early and deep contributions to my love of theory, and who is always the first person I want to share an achievement with.

Finally I would like to thank Yevgeniy Dodis, Guevara Noubir, and Daniel Wichs for their generosity in serving on my thesis committee.
Contents

1 Introduction 1
  1.1 Pseudorandom functions ......................................................... 1
  1.2 Leakage-resilient cryptography ............................................... 5
  1.3 Organization and bibliographic notes ...................................... 7

2 Pseudorandom functions via substitution-permutation networks 8
  2.1 Substitution-permutation networks ......................................... 8
    2.1.1 Security against linear and differential cryptanalysis ............ 9
    2.1.2 Security against degree-exploiting attacks ............................ 13
  2.2 Overview of our candidates ................................................ 15
  2.3 New PRF candidates ............................................................ 19
    2.3.1 Candidate 1 ............................................................. 19
    2.3.2 Candidate 2 ............................................................. 24
    2.3.3 Candidate 3 ............................................................. 28
    2.3.4 Candidate 4 ............................................................. 31
    2.3.5 Candidate 5 ............................................................. 34
  2.4 More related work ............................................................. 35
  2.5 Future directions .............................................................. 37

3 Leakage-resilient circuits via iterated group products 39
  3.1 Overview .................................................................................. 39
    3.1.1 Multiple queries ........................................................... 43
    3.1.2 Only computation leaks ................................................... 44
  3.2 The construction ..................................................................... 46
  3.3 Compressing group products .................................................. 54
    3.3.1 Multi-party protocols ....................................................... 56
    3.3.2 AC^0 ............................................................................ 58
    3.3.3 AC^0 with symmetric gates ................................................ 59
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3.4 TC⁰</td>
<td>62</td>
</tr>
<tr>
<td>3.3.5 NC¹</td>
<td>63</td>
</tr>
<tr>
<td>3.4 Proofs of Corollary 3.1.6 and Theorem 3.1.8</td>
<td>69</td>
</tr>
<tr>
<td>3.5 Multi-query security via secure hardware</td>
<td>74</td>
</tr>
<tr>
<td>3.5.1 Adaptive compression bounds</td>
<td>77</td>
</tr>
<tr>
<td>3.6 Future directions</td>
<td>79</td>
</tr>
<tr>
<td>Bibliography</td>
<td>81</td>
</tr>
<tr>
<td>A Alternate proof of Theorem 3.3.19</td>
<td>89</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Cast broadly, cryptography encompasses many of the most pervasive tasks undertaken by modern computing devices. This field, which studies the means of encoding information so as to be intelligible only to certain parties, has a rich history extending for millenia. With the advent of digital computers in the mid-20th century and the shift to rigorous proof-based methods in the 1970s, cryptography has taken on new and vital roles in computation. Indeed, it is employed today in nearly all forms of electronic communication, and its study additionally raises questions about the nature of computation itself. As such, for reasons both practical and philosophical, we seek to understand the computational complexity of fundamental cryptographic primitives.

With the phrase “cryptographic primitive”, we refer to a basic computational task with well-defined security requirements. Examples include encryption, message authentication, and pseudorandom number generation. In particular, we focus our attention away from complex protocols that may use several such primitives as components.

It has been widely observed that in many cases there is a significant gap between the way that cryptographic primitives are constructed and analyzed by the theory/foundations community, and the way that they are implemented, used, and attacked in practice. In this work we study from two different perspectives the complexity of constructing cryptographic primitives, with an eye towards bridging this gap.

In any study of computational complexity, one must have in mind a model of computation; we will mostly use circuits. In their most basic form, each wire in a circuit carries a bit, and each gate in a circuit computes a simple Boolean function (e.g. And, Or, Not). One can generalize the model by allowing gates that compute more complex functions (e.g. Majority), and by allowing wires that carry values from larger sets (e.g. groups of size > 2).

We now give an overview of this dissertation’s main contributions.

1.1 Pseudorandom functions

One of the most fundamental cryptographic primitives is the task of generating large amounts of random data from a short initial random string (called a “key” or “seed”). Such pseudorandom
data is crucially used in a wide array of cryptographic applications: securing your credit card number when making an online purchase, encrypting messages sent from a base to soldiers in the field, digitally signing a legal document, and countless others.

Constructions implementing this task are in many cases viewed differently between the cryptographic theory community and the practitioner community. In the theory community, these constructions are known as pseudorandom functions (PRF) after the seminal paper of Goldreich, Goldwasser, and Micali [GGM86]. The analogous practical constructions are typically either bounded-input-length hash functions such as the SHA-1 compression function, or block ciphers such as the Advanced Encryption Standard (AES) by Daemen and Rijmen [DR02]. These latter constructions can achieve remarkable efficiency, and their security usually relies on some amount of heuristic analysis arguing that known attacks are unlikely to succeed.

A PRF is a set of efficient functions $f_k : \{0, 1\}^n \rightarrow \{0, 1\}^n$ indexed by a short seed $k$ such that no efficient adversary with oracle access can distinguish a random function in the set (i.e. over a random $k$) from a uniformly random function, except with negligible advantage. In contrast to practical constructions, the security of PRF constructions typically is shown by a reduction to a hardness assumption, such as the difficulty of factoring integers or the existence of one-way functions (OWF), i.e. functions that are easy to compute but hard to invert.

In addition to defining the notion of PRFs, [GGM86] showed how to construct a PRF from any length-doubling pseudorandom generator (PRG). That is, they showed how to construct a PRF if given an efficient function $G : \{0, 1\}^n \rightarrow \{0, 1\}^{2n}$ such that no efficient adversary can distinguish a random output of $G$ from a uniformly random string of length $2n$. Later work by Hästad, Impagliazzo, Levin, and Luby [HILL99] constructed PRGs from any OWF, and an alternative such construction was given by Haitner, Reingold, and Vadhan [HRV10] with improvements by Vadhan and Zheng [VZ12, VZ13]. The combination of these gives a construction of PRFs starting from any OWF (which is the minimal assumption necessary for essentially any cryptographic task).

A different paradigm for constructing PRFs was given by Naor and Reingold [NR04], later with Rosen [NRR02] (cf. [NR99b]). This construction, which is secure assuming either the hardness of factoring or the Decisional Diffie-Hellman assumption, is more efficient than the [GGM86]-based constructions in at least two respects. First, the seed length is quadratic in the input length $n$, whereas the previous constructions require larger seed length. Second, the Naor-Reingold PRF is computable even in the restricted circuit class $TC^0$ of polynomial-size constant-depth circuits consisting of unbounded-fan-in majority gates. In contrast, [GGM86]-
based constructions require circuits of depth at least \( n \), even if the initial PRG is computable by circuits of constant depth. More recently, Banerjee, Peikert, and Rosen [BPR12] use the framework of [NR99b] to construct PRFs whose security is based on the “learning with errors” problem [Reg09] and its variants; at least one of their constructions is also computable in \( \text{TC}^0 \).

Turning to constructions of the practical, bounded-input-length counterparts to PRFs (e.g. AES or the SHA-1 compression function), one finds a troubling gap when compared to the PRF constructions just mentioned. The gap is both quantitative and methodological.

It is quantitative because bounded-input-length constructions often have seed length equal to the input length, while all complexity-based PRF constructions have seed length at least quadratic in the input length. For example, the 128-bit version of AES has seed length equal to the input length. Note that the seed length gives a lower bound on the size of circuits that implement the function.

It is methodological because many modern bounded-input-length hash functions and block ciphers follow the substitution-permutation network (SPN) paradigm. This is for example the case with two of the five finalists for the SHA-3 hash function competition, namely Grøstl [GKM+11] and JH [Wu11], and also AES. An SPN is computed over a number of rounds where each round “confuses” the input by dividing it into bundles and applying a substitution function (S-box) to each bundle, and then “diffuses” the bundles by applying a linear transformation with certain branching properties; the inspiration for this structure comes from the seminal work of Shannon [Sha49]. No piece of this appears to have been used to construct PRFs. Moreover the SPN structure is tailored to resist two general attacks on block ciphers which are largely ignored in the PRF literature, namely linear and differential cryptanalysis.

**Our results.** We take a new step towards closing this gap by giving several candidate PRFs that are inspired by the SPN structure. Each of the many hash functions and block ciphers based on the SPN structure (e.g. those mentioned above) suggests different choices for the parameters, S-boxes, and diffusion matrices. As a first step we choose to follow the design considerations behind the AES block cipher, and particularly its S-box. We do this for two reasons. First, it is a well-documented, widely-used block cipher that has been around for over a decade. Second, the algebraic structure of its S-box lends itself to an asymptotic generalization. We hope that future work will systematically address other available bounded-input-length constructions.

Some of our candidates have better parameters than previous candidates, where by parameters we refer to the seed length and the resources required to compute each function in various
computational models.

1. We first consider an SPN with a random S-box (specified as part of the seed). We prove unconditionally that this resists attacks that run in time less than the seed length. For example we can set the seed length to \( n^c \) and withstand attacks running in time \( n^{c'} \) for sufficiently large \( c \) and \( c' = \Theta(c) \). (Note that being a PRF means that the seed length is \( n^c \) and that the function withstands all attacks running in time \( n^{c'} \) for any \( c' \).)

This result is analogous to that of Luby and Rackoff, who analyzed the Feistel network structure when a certain component is instantiated with a random function, and indeed we prove the same level of security (exponential in the input size of the random function). The techniques used are similar to those in the work by Naor and Reingold [NR99a] that followed Luby and Rackoff’s. To our knowledge this is the first construction of a (provably secure, inefficient) PRF using the SPN structure.

2. Using the AES S-box and a strengthened version of the AES diffusion matrix, we give a candidate computable with Boolean circuits of size \( n \cdot \log^O(1) n \), and in particular with seed length \( O(n \log^2 n) \). We prove that this candidate has exponential security \( 2^{\Omega(n)} \) against linear and differential cryptanalysis by extending a result due to Kang et al. [KHL+01].

3. Again using the AES S-box and a different diffusion matrix, we give a candidate computable with size \( n^{1+\epsilon} \), for any \( \epsilon > 0 \), in the restricted circuit class \( \text{TC}^0 \) of unbounded fan-in majority circuits of constant-depth. The diffusion matrix used here blows up the state to size \( O(n) \), and we output a single bit by taking the inner product of this state with a random string. We prove that this candidate is almost 3-wise independent.

4. We give another single-bit output candidate which uses an extreme setting of the SPN parameters (one round, one S-box, no diffusion matrix). This can be viewed as a slightly modified version of the Even-Mansour cipher [EM97] that uses the AES S-box in place of a random permutation. We prove that this candidate fools all parity tests that look at \( \leq 2^{0.9n} \) outputs.

5. Our final candidate is a straightforward generalization of AES, and may be folklore. We show that it is computable by size \( O(n^2) \), depth \( O(n) \) Boolean circuits, and we further show that for each fixed seed \( k \) it is computable in time \( O(n^2) \) by a single-tape Turing machine with \( O(n^2) \) states. We do not have any proof of security, but the (heuristic) arguments underlying AES’s security also apply to this candidate.
1.2 Leakage-resilient cryptography

We next turn to the construction of cryptographic primitives in a stronger adversarial model than considered in the previous section. The motivation for this study comes from the large number of successful attacks on implementations of cryptographic algorithms that were proven theoretically to be secure. These so-called side-channel attacks exploit properties of the implementation (i.e. the hardware) rather than the algorithm alone. There are numerous such examples; to name just a few: Kocher [Koc96] shows how public keys for the RSA encryption algorithm can be factored by measuring the timing of the algorithm, Kocher et al. [KJJ99] show that the DES encryption algorithm running on a smart card can be broken by measuring the card’s power consumption, and Quisquater and Samyde [QS01] show how electromagnetic radiation can also be used to break DES running on smart cards.

One approach to protecting against such attacks is to improve hardware design, so that e.g. the power consumed by a smart card is independent of the data it processes. Indeed, cryptography traditionally has viewed such attacks as “outside the model”. However for at least the last decade, the cryptography community has itself taken up the challenge of preventing such side-channel attacks, by incorporating these attacks into adversarial models and designing algorithms to provably resist them. This line of work is known as leakage-resilient cryptography.

A general goal in this area is to compile any circuit into a new “shielded” circuit such that any attack exploiting extra information about the circuit’s computation can in fact be carried out just using input/output access (and hence does not succeed under standard hardness assumptions). However, the seminal impossibility result on obfuscation [BGI$^+$12] implies that one cannot shield circuits against an attack that obtains just one extra bit of information about the circuit, if this bit is computed as an arbitrary efficient leakage function of the wires of the circuit. More specifically it is sufficient that the leakage function is powerful enough to evaluate the shielded circuit on its own description. (There is in fact a close connection between obfuscation and leakage resilience, see e.g. the discussion in [Rot12]).

Still, this negative result does not necessarily hinder the scope of a theoretical study of leakage-resilient cryptography, because in practice this extra information is quite difficult to obtain and is typically limited to some simple-to-compute functions such as the Hamming weight of the bits carried on the wires. Thus, it makes sense to focus our attention on attacks where the extra information is obtained from the circuit by evaluating a computationally restricted leakage function.
One line of works considers a model that has become known as “only computation leaks”, after Micali and Reyzin [MR04]. In this model, the compiled circuit is partitioned (by the compiler) into topologically ordered sets of wires, i.e. so that the value of each wire depends only on wires in its set or in sets preceding it. The leakage function then operates separately on each set in the partition. Ishai, Sahai, and Wagner in [ISW03] allow the leakage function to output projections of few of (the values carried on) the wires in each set. Their result is greatly generalized by a series of works [GR10, JV10, DF12, GR12] culminating in the construction by Goldwasser and Rothblum [GR12] which allows any arbitrary function of each set, as long as the function has bounded output length.

In a different direction, Faust et al. [FRR+10] allow leakage functions that are computable by small, bounded-depth circuits with And, Or, and Not gates (AC$^0$). In contrast to the previous setting, here the leakage function accesses all wires simultaneously; we refer to this as the “global” leakage model. In the case of an unbounded number of queries from the adversary – so-called “continual leakage” – the compiled circuits in [FRR+10] use a randomized, inputless gate that outputs a bit vector that is uniform up to having parity = 0. In contrast, the only randomized gates used in [ISW03, GR12] output uniform and independent bits. Gates that produce bits from a distribution other than uniform are referred to in the literature as “secure hardware components”. The use of secure hardware in [FRR+10] was removed by Rothblum [Rot12] at the expense of introducing a computational assumption.

**Our results.** We give a new construction of a leakage-resilient compiler. In the multi-query setting, our construction uses a secure hardware component similar to the one in [FRR+10]. Our construction:

1. is secure against global leakage from log-depth fan-in-2 circuits (NC$^1$), if NC$^1 \neq L$.

2. is candidate to having security in the OCL model; we are not aware of any other construction that is candidate to having security simultaneously in both the global and OCL leakage models.

3. is unconditionally secure against global leakage from natural, well-studied classes of functions that break nearly all previous constructions; for example, it resists leakage from parity and inner product which break [ISW03, FRR+10, DF12, GR12, Rot12].

Our construction derives its security from the hardness of computing iterated products over non-commutative groups. (This differs from all works mentioned above, most of which derive
security from the hardness of computing parity or inner product.) One well-studied such group is $A_n$, the group of even permutations on $n$ points. It has been known since the celebrated work of Barrington [Bar89] that computing products over $A_5$ is complete for NC$^1$, and indeed our use of groups was inspired by his theorem.

However, we require a strictly stronger property of a group $G$ than what is sufficient to support Barrington’s theorem, namely that for every two elements $\alpha, \alpha' \in G$ the leakage function cannot distinguish between vectors with product $= \alpha$ and those with product $= \alpha'$. This class of promise problems had not previously been explicitly considered. We prove lower bounds for this problem in a variety of computational models when $G = A_n$, which yields security against the leakage classes mentioned above. Our construction also motivates new questions in communication complexity; specifically we show that the security of our construction in the OCL model would follow from certain number-on-forehead communication lower bounds for this problem.

### 1.3 Organization and bibliographic notes

In Chapter 2 we construct our PRF candidates using the SPN structure. These results are based on the paper “Substitution-permutation-networks, pseudorandom functions, and natural proofs” [MV12] which is joint work with Emanuele Viola and appeared in the 2012 IACR International Cryptology Conference.

In Chapter 3 we construct our leakage-resilient compiler and show its security by proving lower bounds for computing iterated group products. These results are based on the papers “Shielding circuits with groups” [MV13] and “Iterated group products and leakage-resilience against NC$^1$” [Mil14]. The former is is also joint work with Emanuele Viola and appeared in the 2013 ACM Symposium on the Theory of Computing, and the latter appeared in the 2014 ACM Innovations in Theoretical Computer Science conference. Here these have been merged to streamline the presentation, and we point out that the results of [Mil14] are specifically contained in section 3.3.5.
Chapter 2

Pseudorandom functions via substitution-permutation networks

This chapter is organized as follows. We first review in §2.1 the necessary background on SPNs, and prove our extension of [KHL+01] to \( r > 2 \) rounds, achieving exponential security against linear and differential cryptanalysis. In §2.2 we give an overview of our new PRF candidates, and in §2.3 we give the technical details. In §2.4 we mention for context some related work not discussed in §1.1. We conclude with some future directions in §2.5.

2.1 Substitution-permutation networks

An SPN \( C_k : \{0,1\}^n \to \{0,1\}^n \) is indexed by a key \( k = (k_0, \ldots, k_r) \in (\{0,1\}^n)^{r+1} \), and is specified by the following three parameters and two functions (see Figure 2.1):

- \( r \in \mathbb{N} \), the number of rounds
- \( b \in \mathbb{N} \), the S-box input size
- \( m \in \mathbb{N} \), the number of S-box invocations per round
- \( S : \text{GF}(2^b) \to \text{GF}(2^b) \), the S-box
- \( M : \left(\text{GF}(2^b)\right)^m \to \left(\text{GF}(2^b)\right)^m \), the linear transformation.

The input/output size of \( C_k \) is given by \( n := mb \). Throughout this chapter, we assume a fixed canonical mapping between \( \{0,1\}^b \) and \( \text{GF}(2^b) \).

\( C_k \) is computed over \( r \) rounds. The \( i \)th round (\( 1 \leq i \leq r \)) is computed over three steps:
- (1) \( m \) parallel applications of \( S \);
- (2) application of \( M \) to the entire state;
- (3) XOR of the entire state with the round key \( k_i \).

Note that each round is identical except for step (3). (SPNs are sometimes defined more generally, e.g. by allowing the S-box to vary across rounds or by allowing a more complex interaction with \( k \) than XOR.)

On input \( x \), \( C_k(x) \) gives \( x \oplus k_0 \) as input to the first round; the output of round \( i \) becomes the input to round \( i + 1 \) (for \( 1 \leq i < r \)), and \( C_k(x) \)'s output is the output of the \( r \)th round.
2.1.1 Security against linear and differential cryptanalysis

We now review how the security of an SPN is evaluated against two general attacks on block ciphers: linear and differential cryptanalysis. Resistance to these attacks is typically seen as the main security feature of SPNs. Note that we consider here the basic versions of these attacks, and we leave to future work understanding the resistance of our candidates to more sophisticated attacks (such as those considered by Knudsen [Knu94]).

For both linear and differential cryptanalysis, a crucial property in the security proof is that the linear transformation $M$ has maximal branch number, defined as follows.

**Definition 2.1.1.** Let $M : \mathbb{F}^m \to \mathbb{F}^m$ be a linear transformation acting on vectors over a field $\mathbb{F}$. The branch number of $M$ is

$$\text{Br}(M) = \min_{\alpha \neq 0^m} (w(\alpha) + w(M(\alpha))) \leq m + 1$$

where $w(\cdot)$ denotes the number of non-zero elements.

**Linear cryptanalysis.** Linear cryptanalysis [Mat94] exploits the existence of linear correlations to attack a block cipher $C_k$. For a function $f : \{0,1\}^n \to \{0,1\}^n$ and input/output parities $\Gamma_x, \Gamma_y \in \{0,1\}^n$, define the correlation of $f$ with respect to $\Gamma_x$ and $\Gamma_y$ as

$$\text{Cor}_{\Gamma_x, \Gamma_y}(f) := 2 \cdot \text{Pr}_x[\langle \Gamma_x, x \rangle = \langle \Gamma_y, f(x) \rangle] - 1.$$
For a block cipher $C_k$, the parameter of interest for linear cryptanalysis is

$$p_{LC}(C_k) := \max_{\Gamma_x, \Gamma_y \neq 0} \left( \mathbb{E}_k \left[ \text{Cor}_{\Gamma_x, \Gamma_y} (C_k)^2 \right] \right).$$

Specifically, the attack requires an expected number of plaintext/ciphertext pairs proportional to $1/p_{LC}(C_k)$.

To bound $p_{LC}(C_k)$, the concept of a linear trail is used. Let $\rho_k^i$ denote the $i$th round function of an SPN $C_k$, i.e. $C_k(x) = \rho_k^r(\rho_k^{r-1}(\cdots (\rho_k^1(x \oplus k_0)) \cdots ))$. A linear trail is a vector $\Gamma = (\Gamma_0, \ldots, \Gamma_r) \in \{(0,1)^n\}^{r+1}$, and the correlation of $C_k$ with respect to $\Gamma$ is (cf. [DR02, Eqn. 7.59])

$$\text{Cor}_\Gamma(C_k) := \prod_{i=1}^{r} \text{Cor}_{\Gamma_{i-1}, \Gamma_i}(\rho_k^i).$$

This equation is defined for a fixed key $k$, but in fact for SPNs only the sign of this product is affected by the value of the key [DR02, §7.9.2]. In particular, $\text{Cor}_\Gamma(C_k)^2$ is the same for every key $k$.

For any pair of input/output parities $\Gamma_x, \Gamma_y$, we have the following theorem.

**Theorem 2.1.2** ([DR02], Thm. 7.9.1).

$$\mathbb{E}_k \left[ \text{Cor}_{\Gamma_x, \Gamma_y} (C_k)^2 \right] = \sum_{\Gamma : \Gamma_0 = \Gamma_x, \Gamma_r = \Gamma_y} \text{Cor}_\Gamma(C_k)^2.$$

Note that a naïve evaluation of this sum leads to a useless bound on $p_{LC}$ (i.e. a bound $\geq 1$) due to the large number of vectors $\Gamma$ that have the specified first and final elements.

**Differential cryptanalysis.** Differential cryptanalysis [BS91] attacks a block cipher $C_k$ by exploiting the relationship between the XOR difference of two inputs to $C_k$ and the XOR difference of the corresponding outputs. For a function $f_k : \{0,1\}^n \rightarrow \{0,1\}^n$ parameterized by a key $k$, and input/output differences $\Delta_x, \Delta_y \in \{0,1\}^n$, define the difference propagation probability (DPP) of $f_k$ with respect to $\Delta_x$ and $\Delta_y$ as

$$\text{DPP}_{\Delta_x, \Delta_y}(f_k) := \Pr_{x,k}[f_k(x) \oplus f_k(x \oplus \Delta_x) = \Delta_y].$$

(If $f$ is not parameterized by a key, $k$ is ignored in this definition). For a block cipher $C_k$, the parameter of interest for differential cryptanalysis is

$$p_{DC}(C_k) := \max_{\Delta_x, \Delta_y \neq 0} \left( \text{DPP}_{\Delta_x, \Delta_y}(C_k) \right).$$

Specifically, the attack requires an expected number of plaintext/ciphertext pairs proportional to $1/p_{DC}(C_k)$.
Similarly to how linear trails are used, differential trails are used to bound \( p_{DC}(C_k) \). A differential trail is a vector \( \Delta = (\Delta_0, \ldots, \Delta_r) \in \{0, 1\}^{r+1} \). For any SPN \( C_k \), again let \( \rho_k^i \) denote its \( i \)th round function, and let \( C_k^{(i)}(x) \) denote the output of the \( i \)th round of \( C_k(x) \), with \( C_k^{(0)}(x) := x \oplus k_0 \). That is, for any \( i \leq r \), \( C_k^{(i)}(x) = \rho_k^i(\rho_k^{i-1}((\rho_k^{i-1}(x \oplus k_0)) \cdots)) \).

Then for any \( \Delta_0, \Delta_r \), we have

\[
\text{DPP}_{\Delta_0, \Delta_r}(C_k) = \sum_{\Delta_1, \ldots, \Delta_{r-1}} \Pr_{x,k} \left[ \bigwedge_{i=1}^{r} \left[ \rho_k^i \left( C_k^{(i-1)}(x) \right) \oplus \rho_k^i \left( C_k^{(i-1)}(x \oplus \Delta_0) \right) = \Delta_i \right] \right].
\]

This can be seen by noting that for any fixed values of \( x, k, \Delta_0 \) and \( \Delta_r \), there is at most one tuple \( (\Delta_1, \ldots, \Delta_{r-1}) \) for which the conjunction evaluates to true. To simplify this equation, we use the following two facts.

- The independence of the round keys ensures that, conditioned on two inputs to round \( i \) having XOR difference \( \Delta_{i-1} \), the inputs are uniformly distributed over all pairs with difference \( \Delta_{i-1} \), and are independent of the inputs to all previous rounds.
- XORing the round key does not affect the DPP of a given round. That is, letting \( \rho \) denote the round function without the key XOR, we have \( \text{DPP}_{\Delta_x, \Delta_y}(\rho) = \text{DPP}_{\Delta_x, \Delta_y}(\rho_k^i) \) for all \( i, \Delta_x, \Delta_y \).

Using these facts and an application of the chain rule, we have

\[
\text{DPP}_{\Delta_0, \Delta_r}(C_k) = \sum_{\Delta_1, \ldots, \Delta_{r-1}} \prod_{i=1}^{r} \text{DPP}_{\Delta_{i-1}, \Delta_i}(\rho).
\]

**Proving security.** Our starting point for proving security against linear and differential cryptanalysis is the following theorem due to Kang et al. [KHL+01]. It gives a bound on \( p_{LC} \) and \( p_{DC} \) for 2-round SPNs with maximal branch number.

**Theorem 2.1.3. ([KHL+01], Thms. 5 & 6)** Let \( C_k : \{0, 1\}^n \rightarrow \{0, 1\}^n \) be an SPN with \( r = 2 \) rounds and S-box \( S \). Let \( q := \max_{\Gamma_x, \Gamma_y \neq 0} \left( \text{Cor}_{\Gamma_x, \Gamma_y}(S) \right)^2 \) denote the maximum squared correlation of \( S \), and let \( p := \max_{\Delta_x, \Delta_y \neq 0} \left( \text{DPP}_{\Delta_x, \Delta_y}(S) \right) \) denote the maximum DPP of \( S \). If \( \text{Br}(M) = m + 1 \), then \( p_{LC}(C_k) \leq q^m \) and \( p_{DC}(C_k) \leq p^m \).

For typical S-boxes, such as the one used in AES, one can have \( q = p = 2^{-b+2} \), and so the theorem guarantees security exponential in \( n = mb \). (For completeness we note that one cannot directly apply the above theorem to AES because it is a more complicated SPN.)
In our PRF candidates, we cannot use only two rounds because this leads to other types of attacks (as discussed in the next subsection). Still, we are able to extend Theorem 2.1.3 to \( r > 2 \) rounds as follows.

**Theorem 2.1.4.** Let \( C_k : \{0, 1\}^n \rightarrow \{0, 1\}^n \) be an SPN with \( r = 2\ell \) rounds for some \( \ell \geq 1 \) and S-box \( S \). Let \( q := \max_{x, y \neq 0} \left( \text{Cor}_{\Gamma_x, \Gamma_y} (S)^2 \right) \) denote the maximum squared correlation of \( S \), and let \( p := \max_{\Delta_x, \Delta_y \neq 0} \left( \text{DPP}_{\Delta_x, \Delta_y} (S) \right) \) denote the maximum DPP of \( S \). If \( \text{Br}(M) = m + 1 \),

\[
1. \ p_{\text{LC}}(C_k) \leq q^m \cdot 2^{(\ell-1)n}.
2. \ p_{\text{DC}}(C_k) \leq p^m \cdot 2^{(\ell-1)n}.
\]

Intuitively, the S-box provides security \( q \) (resp. \( p \)) against linear (resp. differential) cryptanalysis, and this security multiplies across “active” S-boxes (instances of \( S \) that are evaluated with a non-zero input). The branch number \( \text{Br}(M) \) guarantees that there exist \( \geq m + 1 \) such active S-boxes in any pair of consecutive rounds, hence the term \( q^m = q^{(r/2)m} \). We note that the factor \( 2^{(\ell-1)n} \) seems to be an artifact of our extension of [KHL+01], and it is open to get a tighter bound on \( p_{\text{LC}} \) and \( p_{\text{DC}} \) for \( r > 2 \) rounds ([KHL+01] only consider \( r = 2 \)). Such an extension has been considered before, for example by Keliher et al. [KMT01] and Cho et al. [CSK+04], but their results only apply in the fixed-parameter setting because they require extensive computer calculation. We are not aware of any other “closed form” bound for \( r > 2 \).

**Proof of Theorem 2.1.4.** We prove part 1; part 2 is essentially identical.

We proceed inductively on \( \ell \). The base case \( \ell = 1 \) is given by Theorem 2.1.3. Fix \( \ell > 1 \), and let \( \Gamma_0, \Gamma_{2\ell} \) be any non-zero input/output parities. Then,

\[
p_{\text{LC}}(C_k) = \sum_{\Gamma=(\Gamma_0, \ldots, \Gamma_{2\ell})} \text{Cor}_{\Gamma}(C_k)^2
\]

\[
= \sum_{\Gamma_1, \ldots, \Gamma_{2\ell-1}} \prod_{i=1}^{2\ell-2} \text{Cor}_{\Gamma_{i-1}, \Gamma_i}(\rho_{k_i})^2
\]

\[
= \sum_{\Gamma_1, \ldots, \Gamma_{2\ell-2}} \prod_{i=1}^{2\ell-2} \text{Cor}_{\Gamma_{i-1}, \Gamma_i}(\rho_{k_i})^2 \sum_{\Gamma_{2\ell-1}} \prod_{i=2\ell-1}^{2\ell} \text{Cor}_{\Gamma_{i-1}, \Gamma_i}(\rho_{k_i})^2
\]

\[
\leq q^m \cdot \sum_{\Gamma_1, \ldots, \Gamma_{2\ell-2}} \prod_{i=1}^{2\ell-2} \text{Cor}_{\Gamma_{i-1}, \Gamma_i}(\rho_{k_i})^2
\]

\[
= q^m \cdot \sum_{\Gamma_{2\ell-2}} \left( \sum_{\Gamma_1, \ldots, \Gamma_{2\ell-3}} \prod_{i=1}^{2\ell-2} \text{Cor}_{\Gamma_{i-1}, \Gamma_i}(\rho_{k_i})^2 \right)
\]

\[
\leq q^m \cdot \sum_{\Gamma_{2\ell-2}} q^{(\ell-1)m} \cdot 2^{(\ell-2)n}
\]

\[
= q^m \cdot 2^{(\ell-1)n}
\]
CHAPTER 2. PSEUDORANDOM FUNCTIONS VIA SUBSTITUTION-PERMUTATION NETWORKS

where (2.1) is by Theorem 2.1.3, (2.2) is by the inductive hypothesis and (2.3) is by the fact that there are $2^n$ choices for $\Gamma_{2^\ell-2}$.

2.1.2 Security against degree-exploiting attacks

While resistance to linear and differential cryptanalysis is the main security feature of the SPN structure (and indeed, “the most important criterion in the design” of AES [DR02, p. 81]), considerations are usually also taken to prevent attacks that would exploit algebraic structure in the cipher. In our candidates 2-5, we adopt essentially the same S-box that is used in AES. This S-box is defined by $S(x) := x^{2^b-2}$ and was chosen to allow the computation to have high degree when considered as a multivariate polynomial over GF(2). Specifically, the use of $x \mapsto x^{2^b-2}$ results in each of $S$’s output bits having (near-maximum) degree $b-1$. Using instead $x \mapsto x^3$ would not diminish resistance to linear and differential cryptanalysis, but it would result in degree (only) 2 [Pie91, Nyb93, Kop11].

We need the degree of each output bit of our candidates (as a multivariate GF(2)-polynomial) to be $\geq \epsilon n$, for some constant $\epsilon$, to resist attacks that exploit the degree of this polynomial. For completeness we present such an attack, showing that a PRF that has degree $o(n)$ cannot have hardness $2^n$. This just follows from the fact that in time $2^n$ one can write down the polynomial representation of $f$ restricted to $\Omega(n)$ input bits. For simplicity, we instead show that any such PRF can be broken in time $2^{O(n)}$. This implies the desired goal, for if we had a PRF $f_k : \{0,1\}^n \rightarrow \{0,1\}$ with hardness $2^n$ we could consider it over $bn$ input bits, note that the degree would still be $o(n) = o(bn)$, and obtain a contradiction.

**Theorem 2.1.5.** Let $F = \{f_k : \{0,1\}^n \rightarrow \{0,1\}\}_k$ be any set of functions such that, for each key $k$, the polynomial representation of $f_k$ over GF(2) has degree $o(n)$. Then there is an adversary that runs in time $\leq 2^{O(n)}$ and distinguishes a random $f_k \in F$ from a random function with advantage $\geq 1 - 2^{-2^{\Omega(n)}}$.

**Proof.** Let $f : \{0,1\}^n \rightarrow \{0,1\}$ be any function, and define the following three values:

- $T_f \in \{0,1\}^{2^n}$ is the truth table of $f$; i.e. $(T_f)_i := f(i)$, identifying a natural number with its binary representation.

- $C_f \in \{0,1\}^{2^n}$ is the coefficient vector of $f$, defined as follows. Fix some ordering on the $2^n$ possible multilinear monomials in $n$ variables. Then, $(C_f)_i = 1$ iff the $i$th monomial appears in the polynomial representation of $f$ over GF(2).
A \in \{0, 1\}^{2^n \times 2^n} is the matrix with rows indexed by the set \(\{0, 1\}^n\) and columns indexed by the set of degree \(\leq n\) multilinear monomials (as with \(C_f\), defined by \(A_{ij} := 1\) iff monomial \(j\) has value 1 under input \(i\).

Note that \(A\) is independent of the function \(f\). Furthermore, \(A\) is invertible because it has full rank, which follows from the fact that any two linear combinations of \(A\)'s columns give the truth tables of two distinct polynomials.

The point is that we can use \(C_f\) to check if the polynomial representation of \(f\) contains a monomial of degree \(\geq n/2\). Clearly this will be false for any \(f_k\) drawn from the PRF, and for a uniformly random function \(F\) we have

\[
\Pr[F\text{ has a monomial of degree } \geq n/2] \geq 1 - 2^{-\binom{n}{n/2}} \geq 1 - 2^{-2^{\Omega(n)}}
\]

which can be seen by viewing \(F\) as being randomly chosen by including each possible monomial independently with probability \(1/2\). Finally, note that \(C_f\) can be computed from the truth table of \(f\) in time \(2^{O(n)}\) as \(C_f = A^{-1} \cdot T_f\).

The only non-linear operation in the entire SPN is the S-box, which for Candidates 2-5 has degree \(b - 1\), and thus the maximum possible degree of each output bit for these candidates is \((b - 1)^r\). Hence we make sure that

\[
b^r \geq n
\]

in each of our candidates. (The distinction between \((b - 1)^r \geq \epsilon n\) and \(b^r \geq n\) is unimportant, as in our candidates we can always increase \(r\) by a constant factor, except in Candidate 4 where we have \(b = n\) and \(r = 1\).) We do not know if \(b^r \geq n\) is sufficient to guarantee degree \(\Omega(n)\), and it is an interesting research direction to understand what restrictions (if any) on the SPN parameters ensure that the function has high degree.

Finally, although a block cipher’s security is often measured against key-recovery attacks, we share many researchers’ viewpoint that distinguishing attacks are the correct model. We also note that there is often an intimate connection between the two types, as many key recovery techniques, including linear and differential cryptanalysis, construct a distinguishing algorithm which is then used to select the correct round keys from a set of potential keys.
CHAPTER 2. PSEUDORANDOM FUNCTIONS VIA SUBSTITUTION-PERMUTATION NETWORKS

2.2 Overview of our candidates

In this section we give an overview of our new PRF candidates. Candidates 1, 2, and 5 output \( n \) bits, while Candidates 3 and 4 output 1 bit. We use \( \mathcal{F}_i \) to refer to the function computing Candidate \( i \). In each candidate, the \((r + 1)n\)-bit round keys are chosen independently and uniformly at random. (Popular constructions typically employ a so-called “key schedule” that generates the round keys from a key of size \( \ll n(r + 1) \).)

**Candidate 1.** Our first candidate \( \mathcal{F}_1 \) is an \( r \)-round SPN with an S-box that is chosen uniformly at random (i.e. specified as part of \( \mathcal{F}_1 \)'s key) from the set of all functions mapping \( \text{GF}(2^b) \) to itself. (Analyzing this candidate when \( S \) is a random permutation is a natural research direction which we do not address here.) The only restriction we make on \( \mathcal{F}_1 \)'s linear transformation \( M \) is that it is invertible and has all entries \( \neq 0 \); we observe that this holds for any \( M \) with maximal branch-number. We show that any adversary \( A \) has small advantage in distinguishing \( \mathcal{F}_1 \) from a random function \( F \).

**Theorem 2.2.1.** If \( A \) makes at most \( q \) total queries to its oracle, then

\[
\left| \Pr_F[A^F = 1] - \Pr_{\mathcal{F}_1}[A^{\mathcal{F}_1} = 1] \right| < O(r^2m^3q^3) \cdot 2^{-b}.
\]

The bound achieved here is similar to that of Luby and Rackoff [LR88] in the sense that it is exponentially small in the size of the random function, with a polynomial loss in the number of queries. (The fact that security degrades with the number of rounds, contrary to what one might expect, seems to be an artifact of the proof.) The proof of this theorem is very similar to that of [NR99a, Thm. 3.2], and proceeds by bounding the collision probability between any two inputs to \( S \) in the final round. However we face an additional hurdle, namely that the inputs to the random function \( S \) in the final round depend on outputs of \( S \) in previous rounds.

By setting \( b = \omega(\log n) \) and \( r = \log n \), we get an inefficient PRF (with security \( n^{\omega(1)} \)). We also note that by setting \( b = c\log n \) for some sufficiently large constant \( c \), \( \mathcal{F}_1 \) is computable in time \( n^{O(c)} \) and has security \( n^{c'} \) for some \( c' = \Omega(c) \).

Finally, note that Theorem 2.2.1 implies corresponding bounds on \( p_{\text{LCS}}(\mathcal{F}_1) \) and \( p_{\text{DSC}}(\mathcal{F}_1) \).

**Candidate 2.** In this candidate we set \( b = \Theta(\log n) \), and we use the AES S-box on \( b \) bits (recall that it maps \( x \mapsto x^{2^b-2} \)). We use a linear transformation \( M \) with maximal branch number, and \( M \) is constructed from an error-correcting code in a similar manner to the linear transformation in AES. (AES’s linear transformation does not have maximal branch number however, a choice
that was made to reduce computation time.) We set the number of rounds $r = \Theta(\log n)$ (observe that $b^r \geq n$).

We prove that Candidate 2 is computable by Boolean circuits of quasilinear-size $\tilde{O}(n) := n \cdot \log^{O(1)} n$. To show this, note that since $r$ is logarithmic it is enough to show how to compute each round with these resources. Moreover, since $b$ is logarithmic, computing the S-boxes comes at little cost.

Our main technical contribution in this candidate is to show how to efficiently compute the linear transformation $M$; specifically, we show that it can be computed with size $\tilde{O}(n)$, for a total circuit size of $r \cdot \left(b^{O(1)} + \tilde{O}(n)\right) = \tilde{O}(n)$. A common method for constructing maximal-branch-number linear transformations is to use the generator matrix $G$ of an $m \rightarrow 2m$ maximum distance separable (MDS) code; specifically, if $G^T = [I \mid A]$, then $M := A$ has maximal branch number. Our method for computing $M$ efficiently has two parts. First, we use a result by Roth and Seroussi [RS85] that if $G$ generates a Reed-Solomon code (which is well-known to be MDS), then $M$ forms a $t \times t$ Cauchy matrix (a type of matrix specified by $O(t)$ elements). We then use a result by Gerasoulis [Ger88] to compute the product of a vector (consisting of bundles of the state) and a Cauchy matrix in quasilinear time; this requires a simple adaptation of the algorithm in [Ger88] to fields of characteristic 2.

By combining Theorem 2.1.4 with a theorem of Nyberg [Nyb93], we show that this candidate has exponential security against linear and differential cryptanalysis.

**Theorem 2.2.2.**

1. $p_{LC}(F_2) \leq 2^{-\Omega(n)}$.
2. $p_{DC}(F_2) \leq 2^{-\Omega(n)}$.

We do not know how to get a candidate computable by circuits of size $O(n)$.

**Candidate 3.** In the previous candidate, the components $S$ and $M$ remain essentially unchanged from AES. In Candidate 3, we also keep $S$ the same (aside from the increase in input/output size), but we modify the linear transformation $M$.

Our observation is that the rationale for using a linear transformation with maximal branch number is just that it allows one to lower bound the number $A$ of so-called “active” S-boxes, which can be defined as follows. Let $C$ be an SPN which uses the identity permutation for $S$ and which has $k_i := 0$ for $0 \leq i \leq r$. Let $w_b : (\{0, 1\}^b)^m \rightarrow \mathbb{N}$ be the function that counts the number of non-zero $b$-bit bundles in its input. Then,

$$A := \min_{0^n \neq x \in \{0,1\}^n} \sum_{i=1}^{r} w_b(\text{state of } C(x) \text{ at the beginning of round } i).$$
This number $\mathcal{A}$ is crucial in evaluating the security of SPNs against linear and differential cryptanalysis (cf. \cite{KHL+01,DR02}). With a simple modification to $M$, we get that a constant fraction of the S-boxes in each round are active. Specifically we use the full generator matrix of an error correcting code with minimum distance $\Omega(n)$, which comes at the expense of expanding the state from $n$ bits to $O(n)$ bits at each round. To counteract the fact that such codes may have some output positions fixed to constant values (leading to a simple distinguishing attack), the computation of Candidate 3 concludes by taking the inner product of the state with a uniform $O(n)$-bit vector that is given as part of the seed. Candidate 3 therefore outputs a single bit.

We take $b = n^\epsilon$ and $r = O(1/\epsilon)$ for arbitrarily small $\epsilon > 0$, and so each round is computable in size

$$\frac{n}{b} \cdot \text{poly}(b) = n^{1+O(\epsilon)},$$

and the whole circuit also in size $n^{1+O(\epsilon)}$.

We further show that Candidate 3 is computable even by TC$^0$ circuits of size $n^{1+O(\epsilon)}$ for any $\epsilon > 0$ (with depth depending on $\epsilon$), cf. \S “The gap between lower bounds and PRF” above. The main technical difficulty in implementing this candidate with the required resources is that the S-box requires computing inversion in a field of size $2^b$ (recall $b = n^{\Omega(1)}$). To implement this in TC$^0$ we note (cf. \cite{HV06}) that inverting the field element $\alpha(x)$ can be accomplished as:

$$\alpha(x)^{2^k-2} = \alpha(x)\sum_{i=1}^{b-1} x^{2^i} = \prod_{i=1}^{b-1} \alpha(x)^{2^i} = \prod_{i=1}^{b-1} \alpha(x^{2^i})$$

where the last equality follows from the fact that we are working in characteristic 2. By hard-wiring the $\leq b$ powers $x, x^2, \ldots, x^{2^{b-1}}$ of $x$ in the circuit, and using the fact that the iterated product of poly$(n)$ field elements is computable by poly$(n)$-size TC$^0$ circuits (see e.g. \cite[Corollary 6.5]{HAB02} and cf. \cite{HV06}), we obtain a TC$^0$ circuit.

Because Candidate 3 deviates somewhat from the SPN structure, we cannot use Theorem 2.1.3, and indeed it is not clear how to define differential cryptanalysis for functions which output only one bit. However, we are able to leverage a technique from differential cryptanalysis to prove that Candidate 3 is almost $3$-wise independent. We were unable to determine if this candidate is $4$-wise independent.

**Definition 2.2.3.** A function $f : \{0,1\}^n \rightarrow \{0,1\}$ parameterized by a key $k$ is $(d,\epsilon)$-wise independent if for any distinct $x_1, \ldots, x_d \in \{0,1\}^n$, the distribution $(f(x_1), \ldots, f(x_d))$ induced by a uniform choice of $k$ is $\epsilon$-close to $U_d$ in statistical distance.
Theorem 2.2.4. $F_3$ is $(3, 2^{-\Omega(n)})$-wise independent.

Finally, we mention that implicit in an assumption that Candidate 3 is indeed hard is the assumption that field inversion cannot be computed by unbounded fan-in constant depth circuits with parity gates $\text{AC}^0[\oplus]$. For otherwise, it can be shown that the whole candidate would be in that class, in contradiction with an algorithm in [RR97, §3.2.1] which distinguishes truth tables of $\text{AC}^0[\oplus]$ functions from random ones in quasipolynomial time. ($M$ can be seen to be a linear operation over $\text{GF}(2)$, hence it can be computed easily with parity gates.) The question of whether field inversion is in $\text{AC}^0[\oplus]$ was raised by Healy and Viola in [HV06]. Their work, and later Kopparty’s [Kop11], do show that several functions related to field inversion are not in $\text{AC}^0[\oplus]$.

**Candidate 4.** In this candidate, we use the extreme setting of parameters $b = n$ and $r = 1$. In other words, Candidate 4 consists of one round, and this round contains only a single S-box (and in particular no linear transformation). This construction can be seen as a concrete instantiation of the Even-Mansour block cipher [EM97], using the AES S-box in place of the random permutation oracle. While this setting does indeed preserve resistance to linear and differential cryptanalysis, we exhibit a simple attack, inspired by Jakobsen and Knudsen [JK01], in which we exploit the algebraic structure to recover the key with just 4 queries.

We then put forth a related candidate $F'_4$ where we only output the Goldreich-Levin bit [GL89]: $F'_4(x) := ((x + k_0)^{2^b - 2}, k_1)$. We prove that this candidate is a $d$-wise small-bias generator with error $d/2^n$ (cf. [NN93, AGHP92]), i.e. that it fools all parity tests that look at $\leq 2^{0.9n}$ outputs.

Theorem 2.2.5. For any choice of $d \leq 2^n$, $F'_4$ is a $d$-wise small-bias generator with error $d/2^n$. That is, for any distinct $a_1, \ldots, a_d \in \{0, 1\}^n$:

$$\left| \Pr_{k_0, k_1} \left[ \sum_{i=1}^{d} F'_4(a_i) = 0 \right] - \frac{1}{2} \right| < \frac{d}{2^n}.$$  

Using Braverman’s result [Bra09] (cf. [Baz09, Raz09]) we obtain that this candidate also fools small-depth $\text{AC}^0$ circuits of any size $w = 2^{n^{o(1)}}$ (that look at only $w$ fixed output bits of the candidate).

This candidate is computable by circuits of quasilinear size $O(n \log^2 n \log \log n)$ using the inversion algorithm in [GvzGPS00]. Using the same ideas for Candidate 3, this candidate is also computable by poly-size $\text{TC}^0$ circuits.
Candidate 5. Our final candidate is a straightforward generalization of AES, and may be folklore. We set \( b = 8 \) as in AES and we again use AES’s S-box. We also use the same linear transformation as in AES (which is slightly different from that of Candidate 2, cf. §2.3.5), except for the necessary increase in the input/output size. We set the number of rounds \( r = n \), and thus the size of the seed is \( |k| = n(n + 1) \).

Candidate 5 is computable by size \( O(n^2) \), depth \( O(n) \) Boolean circuits. For each fixed seed \( k \), Candidate 5 is also computable in time \( O(n^2) \) by a single-tape Turing machine with \( O(n^2) \) states.

We do not know how to get a candidate computable in time \( O(n) \) on a 2-tape Turing machine.

2.3 New PRF candidates

2.3.1 Candidate 1

For our first candidate, we analyze the pseudorandomness of the SPN structure when the S-box is a uniformly random function. The results of this section are of a similar flavor, and use similar techniques, as those of Luby and Rackoff [LR88] and the following work by Naor and Reingold [NR99a]. One notable difference is that we study SPNs as pseudorandom functions, and in particular we do not allow inverse queries to the SPN. (Indeed, if the S-box is not a permutation then the SPN may not be either, in which case inverse queries are not well-defined.) Adapting this proof to handle bidirectional queries is a natural research direction which is not addressed here.

Our analysis in this section holds for SPNs in which the linear transformation \( M \) is invertible and has all entries \( \neq 0 \). We observe that this includes all matrices with maximal branch number.

Claim 1. Let \( M \in (GF(2^b))^{m \times m} \) be any matrix with maximal branch number \( m + 1 \). Then, all entries of \( M \) are non-zero and \( M \) is invertible.

Proof. Assume for contradiction that \( M_{i,j} = 0 \) for some \( i, j \leq m \). Let \( x \in (GF(2^b))^m \) be the vector such that \( x_j = 1 \) and \( x_{j'} = 0 \) for \( j' \neq j \). Then \( (Mx)_i = 0 \), and so \( \text{Br}(M) \leq w(x) + w(Mx) \leq m \).

To see that \( M \) is invertible, simply note that if \( Mx = My \) for \( x \neq y \), then \( M(x + y) = 0^m \). Since \( x + y \neq 0^m \), we would again have \( \text{Br}(M) \leq m \). \( \square \)
For the remainder of this section, fix any invertible \( M \in (\mathbb{GF}(2^b))^{m \times m} \) such that all entries are non-zero. For any function \( S : \mathbb{GF}(2^b) \to \mathbb{GF}(2^b) \) and any set of round keys \((k_0, \ldots, k_{r-1}) \in (\{0,1\}^n)^r\), let \( \mathcal{F}_1 = \mathcal{F}_1(S, k_0, \ldots, k_{r-1}) \) be the r-round SPN on \( n := mb \) bits defined by these components, where the final round consists only of S-boxes (i.e. the final round omits the linear transformation and the key addition).

Let \( A : (\mathbb{N}, \mathbb{N}) \to \{0,1\} \) denote an adversary with oracle access to a function mapping \((\mathbb{GF}(2^b))^m \) to itself; \( A \)'s input is simply \((1^n, 1^b)\) which we omit from now on. We will show that \( A \) has small advantage in distinguishing between the case when its oracle is a uniformly random function \( F \), and when its oracle is \( \mathcal{F}_1 \) for a uniform choice of \((S, k_0, \ldots, k_{r-1})\).

**Theorem 2.2.1.** If \( A \) makes at most \( q \) total queries to its oracle, then

\[
\left| \Pr_{F} [A^F = 1] - \Pr_{\mathcal{F}_1} [A^{\mathcal{F}_1} = 1] \right| < O(r^2 m^3 q^3) \cdot 2^{-b}.
\]

**2.3.1.1 Proof overview**

The proof proceeds in two stages. In the first stage, we consider any set of distinct queries \( x_1, \ldots, x_q \), and we show that there is a low-probability event \( \text{BAD} \) over the choice of \((S, k_0, \ldots, k_{r-1})\) such that, conditioned on \( \neg \text{BAD} \), \( \{\mathcal{F}_1(x_i)\}_{i \leq q} \) is uniformly distributed. Essentially, \( \text{BAD} \) is the event that any two SPN queries induce the same query to some S-box in the final round.

In the second stage, we consider the distribution over transcripts of \( A \)'s interaction with its oracle; we use the results of the first stage in a probability argument to show that the transcripts are distributed nearly identically in either setting, and thus that \( A \)'s distinguishing advantage is small. This framework has been used in a number of other works, e.g. [NR99a, RR00, GR04].

The first stage actually shows that \( \mathcal{F}_1 \) is almost \( q \)-wise independent, or alternatively that it is pseudorandom against adversaries that make \( \leq q \) non-adaptive queries. The technique used in the second stage is a rather generic way of extending the proof to adaptive queries; however we note that it crucially relies on the existence of the event \( \text{BAD} \), and indeed it is not the case that any almost \( q \)-wise independent function is pseudorandom against adversaries making \( q \) adaptive queries. This can be seen for example by considering the distribution over functions \( f : [N] \to [N] \) in which each output is selected uniformly and independently with the restriction that \( f(f(0)) = 0 \). This is almost pairwise-independent, but is trivially distinguishable with two adaptive queries. A different method (that does not give a useful bound in our setting) for obtaining adaptive security from non-adaptive security is given by Hoory et al. [HMMR05, Prop. 3].
We will analyze the first \((r-2)\) rounds of \(F_1\) in a different way from the final 2 rounds, and to this end we define the following two functions. Let \(\rho = \rho(S, k_0, \ldots, k_{r-3})\) compute everything in \(F_1\) before the XOR with \(k_{r-2}\), and let \(\rho' = \rho'(S, k_{r-2}, k_{r-1})\) compute the remainder of \(F_1\). So, \(F_1(x) = \rho'(\rho(x))\). As handling \(\rho'\) will be the more involved part of the analysis, we note that it can be written as

\[
\rho'(x) := S^* \left( M \cdot S^*(x + k_{r-2}) + k_{r-1} \right)
\]

where for any \(x = x^{(1)} \cdots x^{(m)}\) we define \(S^*(x) := (S(x^{(1)}), \ldots, S(x^{(m)}))\).

### 2.3.1.2 Stage 1

Fix distinct \(x_1, \ldots, x_q \in (\text{GF}(2^b))^m\). We view \((S, k_0, \ldots, k_{r-1})\) being chosen as follows:

1. Uniformly choose \(k_0, \ldots, k_{r-3}\).

2. Run the computation of \(\rho(x_i)\) for all \(i \leq q\), and each time the S-box is evaluated on a previously-unseen input, choose the output uniformly at random. Let \(H \subseteq \text{GF}(2^b)\) be the set of at most \(qm(r-2)\) S-inputs whose output is determined after this step.

3. Uniformly choose \(k_{r-2}\).

4. Uniformly choose the output of \(S\) on each block of each \((\rho(x_i) + k_{r-2})\) whose output is not already determined.

5. Uniformly choose \(k_{r-1}\).

6. Uniformly choose the output of \(S\) on all remaining inputs.

It is clear that, for any \(x_1, \ldots, x_q\), this distribution is uniform. Our analysis will use the state of the SPN’s computation immediately before the final round of S-boxes, and we denote this state for the \(i\)th query by

\[
z_i := M \cdot S^*(\rho(x_i) + k_{r-2}) + k_{r-1}.
\]

We now define the event \(\text{BAD}\). Informally \(\text{BAD}\) holds if, after step 5, any of the S-inputs that need to be evaluated (i.e. the blocks of the \(z_i\)) collide either with each other or with one of the inputs selected in steps 2 and 4. To reduce notation, we use the following definition.

**Definition 2.3.1.** Let \(x, y \in (\text{GF}(2^b))^m\), and denote \(x = x^{(1)} \cdots x^{(m)}\) and \(y = y^{(1)} \cdots y^{(m)}\). Then, we say that \(x\) and \(y\) collide if \(\exists \ell, \ell' : x^{(\ell)} = y^{(\ell')}\). Further, for any \(T \subseteq \text{GF}(2^b)\), we
say that $x$ and $T$ collide if $\exists \ell \leq m, t \in T : x^{(\ell)} = t$. Finally, we say that $x$ self-collides if $\exists \ell \neq \ell' : x^{(\ell)} = x^{(\ell')}$. 

Now, let $\text{BAD} = \text{BAD}(x_1, \ldots, x_q)$ be the set of all $(S, k_0, \ldots, k_{r-1})$ such that at least one of the following holds:

(a) $\exists h, h' \in H : S(h) = S(h')$.
(b) $\exists i < q : z_i \text{ and } H \text{ collide}$.
(c) $\exists i, i' \leq q : z_i \text{ and } (\rho(x_i) + k_{r-2}) \text{ collide}$.
(d) $\exists i \leq q : z_i \text{ self-collides}$.
(e) $\exists i \neq i' \leq q : z_i \text{ and } z_i' \text{ collide}$.

It is crucial for us that determining whether $\text{BAD}$ holds can be checked after step 5 in choosing $(S, k_0, \ldots, k_{r-1})$.

We now prove two lemmas showing that $\text{BAD}$ occurs with low probability, and that the query answers are uniformly distributed when conditioned on $\neg \text{BAD}$. In the remainder of this subsection, we will simply use $\text{BAD}$ to mean $(S, k_0, \ldots, k_{r-1}) \in \text{BAD}$.

**Lemma 2.3.2.** $\Pr_{S,k_0,\ldots,k_{r-1}}[\text{BAD}] < O(r^2 m^3 q^3) \cdot 2^{-b}$.

**Proof.** We start by bounding the probability of items (a)-(d) individually.

First, we have $\Pr_{S,k_0,\ldots,k_{r-3}}[(a)] < (qm(r - 2))^2 \cdot 2^{-b}$ by a union bound over all pairs of S-box instances in the first $r - 2$ rounds.

We analyze (b)-(e) starting after step 2, so for these let $k_0, \ldots, k_{r-3}, H$, and $S(H)$ be fixed arbitrarily.

Fix any $k_{r-2}$ and the outputs of $S$ on the blocks of $(\rho(x_i) + k_{r-2})$ for all $i$, which fixes $\tilde{z}_i := M \cdot S^*(\rho(x_i) + k_{r-2})$; note that $z_i = \tilde{z}_i + k_{r-1}$. Then, $\Pr_{k_{r-1}}[(b)] \leq qm \cdot qm(r - 2) \cdot 2^{-b}$ by a union bound over each block of each $\tilde{z}_i$ and each element of $H$.

By the same argument as for (b), $\Pr_{k_{r-1}}[(c)] \leq (qm)^2 \cdot 2^{-b}$, where the union bound is now over each block of each $z_i$ and each block of each $(\rho(x_i) + k_{r-2})$.

Let the $\tilde{z}_i$ be defined as above; then for each $i$, $\Pr_{k_{r-1}}[(z_i = \tilde{z}_i + k_{r-1}) \text{ self collides}] < m^2 \cdot 2^{-b}$ by a union bound over pairs of blocks. So, $\Pr_{k_{r-1}}[(d)] < qm^2 \cdot 2^{-b}$.

We will now bound $\Pr[(e)]|\neg(a)]$. Note that $\neg(a)$ implies that each component of $\rho$ is injective and thus that each $\rho(x_i)$ is distinct (this is where we use $M$’s invertibility).
Fix any $i \neq i' \leq q$ and $\ell, \ell' \leq m$. We will show that $\Pr[z^{(\ell)}_i = z^{(\ell')}_{i'} \mid \neg(a)] < O(rmq) \cdot 2^{-b}$, and then a union bound over $i, i', \ell, \ell'$ gives $\Pr[(e) \mid \neg(a)] < O(rm^2q^3) \cdot 2^{-b}$. (We remark that the non-trivial case is when $\ell = \ell'$, i.e. when comparing the same final-round S-box for distinct $x_i, x_{i'}$, because in this case the same block of $k_{r-1}$ affects both $S$-inputs. If $\ell \neq \ell'$ then one can proceed similarly to (a)-(d), but the following works for either case.)

From the definition of $z_i$ we have $z^{(\ell)}_i = z^{(\ell')}_{i'}$ iff
\[
k_{r-1}^{(\ell)} + \sum_{s=1}^{m} M_{\ell,s} \cdot S(\rho(x_i)^{(s)} + k_{r-2}^{(s)}) = k_{r-1}^{(\ell')} + \sum_{s=1}^{m} M_{\ell',s} \cdot S(\rho(x_{i'})^{(s)} + k_{r-2}^{(s)}).
\] (2.4)

Let $t$ be such that $\rho(x_i)^{(t)} \neq \rho(x_{i'})^{(t)}$, which must exist because $\rho(x_i) \neq \rho(x_{i'})$. Arbitrarily fix $k_{r-2}^{(s)}$ for all $s \neq t$, the outputs of $S$ on the input set $I := \{(\rho(x_i) + k_{r-2})^{(s)}, (\rho(x_{i'}) + k_{r-2})^{(s)}\}_{s \neq t}$, and $k_{r-1}^{(\ell)}$ and $k_{r-1}^{(\ell')}$. This fixes an $\alpha \in \text{GF}(2^b)$ such that (2.4) holds iff
\[
M_{\ell,t} \cdot S(\rho(x_i)^{(t)} + k_{r-2}^{(t)}) + M_{\ell',t} \cdot S(\rho(x_{i'})^{(t)} + k_{r-2}^{(t)}) = \alpha.
\] (2.5)

If neither $\rho(x_i)^{(t)} + k_{r-2}^{(t)}$ nor $\rho(x_{i'})^{(t)} + k_{r-2}^{(t)}$ are in $(I \cup H)$, then (2.5) holds with probability $= 2^{-b}$ over the choice of $S$ on these two inputs (this is where we use the fact that all entries of $M$ are non-zero). Further, these two inputs fall inside $(I \cup H)$ with probability $\leq 2 \cdot (qm(r - 2) + 2(m - 1)) \cdot 2^{-b}$ over the choice of $k_{r-2}^{(t)}$, by a union bound over the elements of $(I \cup H)$. Thus $\Pr[z^{(\ell)}_i = z^{(\ell')}_{i'} \mid \neg(a)] = O(rmq) \cdot 2^{-b}$ as promised.

Finally, we have $\Pr[(a) \lor \cdots \lor (e)] \leq \Pr[(a)] + \cdots + \Pr[(d)] + \Pr[(e) \mid \neg(a)] < O(r^2m^3q^3) \cdot 2^{-b}$. \qed

**Lemma 2.3.3.** For any distinct $x_1, \ldots, x_q$ and any $y_1, \ldots, y_q$:
\[
\Pr_{S, k_0, \ldots, k_{r-1}} \left[ \forall i \leq q : F_1(x_i) = y_i \mid \neg\text{BAD} \right] = 2^{-qm b}.
\]

**Proof.** After running steps 1 through 5 in the process of choosing $(S, k_0, \ldots, k_{r-1})$, if we condition on $\neg\text{BAD}$ then the $qm$ elements of the set $\{z^{(\ell)}_i\}_{i,\ell}$ are unique and were not used as inputs to $S$ in steps 2 or 4. Thus, each element has a $2^{-b}$ probability (independent from the other elements) of being mapped by $S$ to the corresponding output (i.e. a block of a $y_i$), and the lemma follows. \qed

### 2.3.1.3 Stage 2

We now show that even adversaries that make adaptive queries have small distinguishing advantage, i.e. we prove Theorem 2.2.1. We make the standard assumption that the adversary $A$ is deterministic, computationally unbounded, and never queries an oracle twice with the same input.
To prove Theorem 2.2.1, we extend the results of the previous section by considering the distribution over transcripts of $A$’s interaction with its oracles. A transcript is a sequence $\sigma = [(x_1, y_1), \ldots, (x_q, y_q)]$ that contains the query/answer pairs arising from $A$’s interaction with its oracle. We use $T_F$ to denote the transcript of $A^F$, and we use $A(\sigma)$ to denote $A$’s output after seeing transcript $\sigma$. (So note for instance that $\Pr_F[A^F = 1]$ and $\Pr_F[A(T_F) = 1]$ are semantically equivalent.)

Because $A$ is deterministic, there is a deterministic function $Q_A$ that determines its next query from the partial transcript so far. For a transcript $\sigma$, denote its prefixes by $\sigma_i := [(x_1, y_1), \ldots, (x_i, y_i)]$. We say a transcript $\sigma$ is possible for $A$ if for all $i < q$: $Q_A(\sigma_i) = x_{i+1}$. Clearly for any impossible transcript $\sigma$, $\Pr[T_F = \sigma] = 0$ regardless of the distribution from which $F$ is chosen. Also note that the assumption that $A$ never makes the same query twice implies that in any possible transcript, $x_i \neq x_j$ for all $i \neq j$.

We now prove Theorem 2.2.1 with a probability argument similar to [NR99a, Thm. 3.2].

Proof of Theorem 2.2.1. Let $\Gamma$ be the set of possible transcripts such that $A(\sigma) = 1 \iff \sigma \in \Gamma$. Then,

$$
\left| \Pr_F[A^F = 1] - \Pr_F[A^F_1 = 1] \right| \\
= \sum_{\sigma \in \Gamma} \left( \Pr_F[T_F = \sigma] - \Pr_{S,k_0,\ldots,k_{r-1}}[T_{F_1} = \sigma] \right) \\
\leq \sum_{\sigma \in \Gamma} \Pr_{S,k_0,\ldots,k_{r-1}}[\text{BAD}] \cdot \left( \Pr_F[T_F = \sigma] - \Pr_{S,k_0,\ldots,k_{r-1}}[T_{F_1} = \sigma | \text{BAD}] \right) \\
+ \sum_{\sigma \in \Gamma} \Pr_{S,k_0,\ldots,k_{r-1}}[\neg\text{BAD}] \cdot \left( \Pr_F[T_F = \sigma] - \Pr_{S,k_0,\ldots,k_{r-1}}[T_{F_1} = \sigma | \neg\text{BAD}] \right). \quad (2.6)
$$

(2.7) implies that $2qmb$ for any possible transcript $\sigma$. We rewrite (2.6) as

$$
\left| \sum_{\sigma \in \Gamma} \left( \Pr_F[\text{BAD}] \cdot \Pr_F[T_F = \sigma] \right) - \sum_{\sigma \in \Gamma} \left( \Pr_{S,k_0,\ldots,k_{r-1}}[\text{BAD}] \cdot \Pr_{S,k_0,\ldots,k_{r-1}}[T_{F_1} = \sigma | \text{BAD}] \right) \right|.
$$

Each of the two summations is bounded by $\alpha := \max_{\sigma \in \Gamma} \left( \Pr_{S,k_0,\ldots,k_{r-1}}[\text{BAD}] \right)$, since each is a convex combination of numbers that are bounded by $\alpha$. Thus, the absolute value of their difference is bounded by $\alpha$ as well, and $\alpha < O(r^2m^3q^3) \cdot 2^{-b}$ by Lemma 2.3.2. \qed
CHAPTER 2. PSEUDORANDOM FUNCTIONS VIA SUBSTITUTION-PERMUTATION NETWORKS

2.3.2 Candidate 2

Our next candidate PRF $F_2 : \{0,1\}^n \rightarrow \{0,1\}^n$ is parameterized by a key $k$ of length $O(n \log^2 n)$ and is computable by circuits of size $\leq n \log^{O(1)} n$. We will show that it has security $2^{-\Omega(n)}$ against linear and differential cryptanalysis via Theorem 2.1.4. The SPN defining $F_2$ closely follows AES.

Definition of $F_2$. $F_2$ is an SPN as defined in §2.1; our parameter choices are as follows. For any $b \in \mathbb{N}$ let $m := 2^{b-1}$, $r := \lceil b/10 \rceil$ and $n := mb$.

For the S-box, we use essentially the same function used in AES. Namely, $S : \text{GF}(2^b) \rightarrow \text{GF}(2^b)$ is defined by $S(x) := x^{2^{b-2}}$. Note that $x \mapsto x^{2^{b-2}}$ is simply inversion in $\text{GF}(2^b)$ with $0^{-1} := 0$. The bounds on $p_{LC}$ and $p_{DC}$ from Theorems 2.1.3 and 2.1.4 are stated in terms of bounds on the correlation and the DPP, respectively, of the S-box. The results of Nyberg [Nyb93] and the references therein establish these bounds, stated in following theorem.

Theorem 2.3.4. Let $S : \text{GF}(2^b) \rightarrow \text{GF}(2^b)$ be defined by $S(x) := x^{2^{b-2}}$. Then:

1. $\max_{\Gamma_x, \Gamma_y \neq 0} \left( \text{Cor}_{\Gamma_x, \Gamma_y}(S) \right)^2 \leq 2^{b-2}$. \footnote{[Nyb93] actually bounds a related quantity known as the non-linearity of $S$, but it translates directly to the stated result.}

2. $\max_{\Delta_x, \Delta_y \neq 0} \left( \text{DPP}_{\Delta_x, \Delta_y}(S) \right) \leq 2^{b-2}$.

For the linear transformation $M : \text{GF}(2^b)^m \rightarrow \text{GF}(2^b)^m$, the crucial property is that it has maximal branch number $\text{Br}(M) = m + 1$. Let $G$ be the $2m \times m$ generator matrix of a Reed-Solomon code over $\text{GF}(2^b)$. (Note that $2^b \geq 2m$ is sufficient to guarantee the existence of such a code.) Take $G$ to be in reduced echelon form, i.e. take $G^T = [I \mid M]$ where $I$ is the $m \times m$ identity matrix. Then, because $G$ generates a maximum-distance-separable (MDS) code, it can be verified that the operation defined by left multiplication with $M$ has branch number $m + 1$. This use of MDS codes to create maximal-branch-number transformations is widespread, and dates at least to [Dae95].

Security of $F_2$. The security of $F_2$ is given by the following theorem (restated).

Theorem 2.2.2. \begin{itemize} \item[1.] $p_{LC}(F_2) \leq 2^{-\Omega(n)}$. \item[2.] $p_{DC}(F_2) \leq 2^{-\Omega(n)}$. \end{itemize}

We note that this security is “as good” as what is available for AES, i.e. AES on 128-bit inputs is believed to have security $\approx 2^{128}$. (In fact, AES’s security relies on some heuristic arguments which we avoid.) Furthermore, resistance to linear and differential cryptanalysis is essentially the only type of security currently available for SPNs.
Given the choices of \( b, r, \) and \( m \), Theorem 2.2.2 follows immediately from Theorem 2.3.4 and Theorem 2.1.4, restated below.

**Theorem 2.1.4.** Let \( C_k : \{0,1\}^n \rightarrow \{0,1\}^n \) be an SPN with \( r = 2\ell \) rounds for some \( \ell \geq 1 \) and S-box \( S \). Let \( q := \max_{\Gamma_x, \Gamma_y \neq 0} \left( \text{Cor}_{\Gamma_x, \Gamma_y} \left( S \right)^2 \right) \) denote the maximum squared correlation of \( S \), and let \( p := \max_{\Delta_x, \Delta_y \neq 0} \left( \text{DPP}_{\Delta_x, \Delta_y} \left( S \right) \right) \) denote the maximum DPP of \( S \). If \( Br(M) = m + 1 \),

1. \( p_{\text{LC}}(C_k) \leq q^m \cdot 2^{(\ell-1)n} \).
2. \( p_{\text{DC}}(C_k) \leq p^m \cdot 2^{(\ell-1)n} \).

By varying the constant 10 in \( r := \left\lceil \frac{b}{10} \right\rceil \), \( p_{\text{LC}}(F_2) \) and \( p_{\text{DC}}(F_2) \) can be bounded by \( 2^{-(1-\epsilon)n} \) for any fixed \( \epsilon > 0 \).

We also note that a bound of \( p_{\text{LC}}(C_k) \leq 2^{-\Omega(n)} \) incorporates the same bound on each Fourier coefficient of each output bit of \( C_k \). In turn, this implies that each output bit depends on \( \Omega(n) \) input bits. (Otherwise it can be verified that it would have too large a Fourier coefficient by Parseval’s identity.)

**Efficiency of \( F_2 \).** We now explain how to compute \( F_2 \) in quasilinear size. The “tricky” component is multiplication by \( M \). Roth and Seroussi [RS85, Theorem 1] show that when the Reed-Solomon matrix \( G \) is put into reduced echelon form, i.e. when \( G^T = [I | M] \), then \( M \) is a generalized Cauchy matrix, defined as follows.

**Definition 2.3.5.** Let \( \mathbb{F} \) be any field of characteristic 2. A matrix \( C \in \mathbb{F}^{m \times m} \) is a **Cauchy matrix** if there exist \( 2m \) distinct values \( \alpha_1, \ldots, \alpha_m, \beta_1, \ldots, \beta_m \in \mathbb{F} \) such that \( C_{i,j} = (\alpha_i + \beta_j)^{-1} \). Furthermore, a matrix \( M \in \mathbb{F}^{m \times m} \) is a **generalized Cauchy matrix** if it can be written as \( M = BCD \), where \( C \) is a Cauchy matrix and \( B, D \in \mathbb{F}^{m \times m} \) satisfy \( B_{i,j} = 0 \Leftrightarrow i \neq j \) and \( D_{i,j} = 0 \Leftrightarrow i \neq j \).

Gerasoulis [Ger88] shows that multiplication of a vector by an \( m \times m \) Cauchy matrix can be done with \( \widetilde{O}(m) \) operations when the underlying field is \( \mathbb{C} \). (Multiplication with \( B \) and \( D \) in the above definition can be done with \( O(m) \) operations, so we will focus on multiplication by \( C \).)

This algorithm can also be made to work over \( GF(2^b) \), as we now show. We stress that we are using the same algorithm from [Ger88]; the purpose here is to show that it works over \( GF(2^b) \).

**Theorem 2.3.6.** Let \( C \in GF(2^b)^{m \times m} \) be a Cauchy matrix defined by the (distinct) elements \( \{\alpha_j, \beta_j\}_{j \in [m]} \). Then, given any vector \( z \in GF(2^b)^m \), the product \( C \cdot z \) can be computed with \( O(m \cdot \log^2 m \cdot \log \log m) \) operations over \( GF(2^b) \).
Proof. Define the following polynomial.

$$f(x) := \sum_{j=1}^{m} z_j (x + \beta_j)^{2^b - 2}$$

Then we have $C \cdot z = (f(\alpha_1), \ldots, f(\alpha_m))$, and so it suffices to evaluate $f$ at the points $\{\alpha_i\}_i$. Now define the following three polynomials.

$$g(x) := \prod_{j=1}^{m} (x + \beta_j)$$

$$h(x) := \sum_{i=1}^{m} \left[ z_i (x + \beta_i)^{2^b - 1} \cdot \prod_{j \neq i} (x + \beta_j) \right]$$

$$h_*(x) := \sum_{i=1}^{m} z_i \prod_{j \neq i} (x + \beta_j)$$

Then we have $f(x) = h(x)/g(x)$ as formal polynomials. Furthermore, for any $y \notin \{\beta_j\}_j$ we have $h(y) = h_*(y)$, using the identity $y^{2^b - 2} = 1$ valid for any $y \neq 0$. Since our goal is to evaluate $f(\alpha_i)$ for all $i$, this is now seen to be equivalent to evaluating $h_*(\alpha_i)/g(\alpha_i)$ because $\alpha_i \neq \beta_j$ for all $i, j$.

Notice that, for each $\beta_j$, we have $h_*(\beta_j) = z_j \cdot g'(\beta_j)$, where $g'(x) = \sum_{i \in [m]} \prod_{j \neq i} (x + \beta_j)$ is the derivative of $g$. So, another way to view $h_*(x)$ is that it is the unique degree $\leq m - 1$ polynomial interpolating the points $\{(\beta_j, z_j \cdot g'(\beta_j))\}_{j \in [m]}$. The algorithm is now the following:

1. Compute $g(x)$ and $g'(x)$ in coefficient form.
2. Evaluate $g'(\beta_j)$ for each $\beta_j$.
3. Compute all values of $z_j \cdot g'(\beta_j)$.
4. Interpolate the points $\{(\beta_j, z_j \cdot g'(\beta_j))\}$ to obtain $h_*(x)$ in coefficient form.
5. Evaluate both $g(\alpha_j)$ and $h_*(\alpha_j)$ for each $\alpha_j$.
6. Compute each value of $f(\alpha_j) = h_*(\alpha_j)/g(\alpha_j)$.

We note that steps 1 and 2 do not involve the vector $z$ and thus can be pre-processed, and that steps 3 and 6 can easily be done with $m$ operations over $\text{GF}(2^b)$ each. For the remaining steps, we use the following results which can be found in (for example) [vzGG03, Ch. 10] and which hold for any commutative ring with unity $R$.

**Theorem 2.3.7** ([vzGG03], Corollary 10.8). Evaluation of a polynomial in $R[x]$ of degree $< m$ at $m$ points can be done with $O(m \cdot \log^2 m \cdot \log \log m)$ operations in $R$. 

Theorem 2.3.8 ([vzGG03], Corollary 10.12). Given $m$ distinct values $u_1, \ldots, u_m \in R$ and $m$ arbitrary values $v_1, \ldots, v_m \in R$, the unique polynomial in $R[x]$ of degree $< m$ which interpolates $\{(u_i, v_i)\}_i$ can be computed in coefficient form with $O(m \cdot \log^2 m \cdot \log \log m)$ operations in $R$.

As a result, steps 4 and 5, and thus the entire multiplication by $C$, can be performed with the stated number of operations in GF($2^b$).

One round of $F_2$ consists of the following three steps:

1. $m$ parallel instances of exponentiation in GF($2^b$) (i.e. $x \mapsto x^{2^b-2}$).
2. One instance of multiplication by $M \in$ GF($2^b$)$^{m \times m}$.
3. One instance of the round key xor.

Because finite field arithmetic and affine transformations are computable by polynomial size circuits, step (1) can be computed by a circuit with at most $m \cdot b^{O(1)}$ wires. For step (2), we have size at most $m \cdot \log^3 m \cdot b^{O(1)}$ by Theorem 2.3.6. Step (3) can clearly be done with $O(mb)$ wires. Thus, given the choices of $m$, $b$, and $r$ above, the $r$ rounds of $F_2 : \{0, 1\}^n \to \{0, 1\}^n$ are computable by a circuit of size $n \cdot \log^{O(1)} n$, and the key size is $|k| = mbr = O(n \log^2 n)$.

2.3.3 Candidate 3

In this section we define a candidate PRF $F_3 : \{0, 1\}^n \to \{0, 1\}$ parameterized by a key of length $O(n)$ and computable by TC$^0$ circuits of size $O(n^{1+\epsilon})$ for arbitrarily small constant $\epsilon > 0$. The construction is again inspired by the SPN structure, and the S-box $S$ is defined identically to that of $F_2$, but the linear transformation $M$ takes a somewhat different form.

The linear transformation $M$. $M$ is constructed using a good error correcting code as before; specifically, we use codes given by the following theorem, which follows from [GHK+13, Theorem 1].

Theorem 2.3.9. For any constant $\epsilon > 0$, there exist constants $c, \delta > 0$ such that for sufficiently large $\ell$, there exists a linear code $C_\epsilon : \{0, 1\}^{\ell/c} \to \{0, 1\}^\ell$ which has distance $\geq \delta \cdot \ell$ and is computable by a TC$^0$ circuit of size $O(\ell^{1+\epsilon})$.

Rather than using a portion of $C_\epsilon$’s generator matrix as with $F_2$ however, $M$ consists of the entire matrix that generates $C_\epsilon$. As a result, the internal state grows by a factor of $c$ during each round, and thus the $M$ used at round $i$ will be a $c^i n \times c^{i-1} n$ matrix.
To see the advantage that this has over the previous choice of $M$, consider an input vector to $M$ in which all $b$-bit bundles are non-zero. If we use only the fact that $M$ has maximal branch number (Definition 2.1.1), then we are only guaranteed that $M$’s output will have one non-zero bundle. However if we instead take $M = C_\epsilon$, then we are guaranteed that at least $\delta \cdot m$ of the output bundles will be non-zero (where $n = mb$), even if all input bundles were non-zero.

**Definition of $F_3$.** Let $m, b, r \in \mathbb{N}$ be arbitrary for now, and set $n := mb$. Fix any $\epsilon > 0$; let $c, \delta, C_\epsilon$ be given by Theorem 2.3.9, and for $1 \leq i \leq r$ let $M^{(i)}$ be the matrix that generates $C_\epsilon$ when $\ell := c^i n$ in Theorem 2.3.9. Let $k = (k_0, \ldots, k_{r+1})$ denote the key of $F_3$, where $|k_i| = c^i n$ for $0 \leq i \leq r$ and $|k_{r+1}| = |k_r| = c^r n$.

$F_3 : \{0, 1\}^n \to \{0, 1\}$ is computed over $r$ rounds. For $1 \leq i \leq r$, round $i$ maps $\{0, 1\}^{c^{i-1} n}$ to $\{0, 1\}^{c^i n}$, and is computed over three steps: (1) $c^{i-1} m$ parallel applications of $S$; (2) application of $M^{(i)}$ to the entire state; (3) XOR of the entire state with the round key $k_i$. (Note that this is exactly the same structure as an SPN, except that $M$ is no longer a permutation, though it is still injective.)

On input $x$, $F_3(x)$ gives $x \oplus k_0$ as input to the first round; the output of round $i$ becomes the input to round $i + 1$ (for $1 \leq i < r$), and $F_3(x)$ outputs $(y, k_{r+1}) \in \{0, 1\}$ where $y$ denotes the output of round $r$. (The inner product with $k_{r+1}$ is the second way in which this candidate deviates from the SPN structure.)

**Efficiency of $F_3$.** We now show that $F_3$ can be computed by $\text{TC}^0$ circuits of size $O(n^{1+\epsilon})$. For $1 \leq i \leq r$, round $i$ consists of the following:

1. $c^{i-1} m$ parallel instances of exponentiation in $GF(2^b)$ (i.e. $x \to x^{2^b-2}$).
2. One instance of multiplication by $M^{(i)}$.
3. One instance of the round key XOR.

Step (1) is computable by a $\text{TC}^0$ circuit of size $c^{i-1} \cdot m \cdot b^{O(1)}$, using the technique described in §2.2. Step (2) is computable with size $O(c^{1+\epsilon} \cdot n^{1+\epsilon})$ by Theorem 2.3.9. Step (3) can be computed with size $O(c^r n)$. The final inner product with $k_{r+1}$ can be computed with size $O(c^r n)$.

Putting it together, there exists a constant $\kappa$ such that the entire function can be computed by a threshold circuit of size $O \left( r \cdot (c^\kappa mb^\kappa + c^{(1+\epsilon)r \cdot n^{1+\epsilon}}) \right)$ and depth $O(r)$. Let $b \in \mathbb{N}$ be sufficiently large, and set $m := \lceil b^{(\kappa - \epsilon - 1)/\epsilon} \rceil$ and $r := \lceil \kappa/\epsilon \rceil$. With $n := mb$, this ensures that $mb^\kappa \leq n^{1+\epsilon}$
(and also that \( b^r \geq n \)). Thus, the entire function is indeed computable by a \( \text{TC}^0 \) circuit of size \( O(n^{1+\epsilon}) \), where both the depth and the hidden constant depend on \( \epsilon \).

**Security of \( F_3 \).** Here we are able to leverage techniques from differential cryptanalysis to prove that \( F_3 \) is almost 3-wise independent. Specifically, the proof below uses a technique from Nyberg’s proof of Theorem 2.3.4 [Nyb93].

**Theorem 2.2.4.** \( F_3 \) is \((3, 2^{-\Omega(n)})\)-wise independent.

**Proof.** We will show that \( F_3 \) is a 3-wise \( 2^{-\Omega(n)} \)-bias generator, i.e. that for every \( d \leq 3 \) and any distinct \( x_1, \ldots, x_d \in \{0,1\}^n \), \( |\Pr_k [\sum_i F_3(x_i) = 0] - 1/2| < 2^{-\Omega(n)} \). By a well-known fact (cf. [AGHP92, Lemma 1]) this implies the theorem.

For any input \( x \), let \( F_3^3(x) \in \{0,1\}^{c_n} \) denote the state just before the final inner product; that is, \( F_3(x) = (F_3^3(x), k_{r+1}) \). Then, \( \sum_i F_3(x_i) = (\sum_i F_3^3(x_i), k_{r+1}) \). So, we will show that for any \( d \leq 3 \) and any distinct \( x_1, \ldots, x_d \in \{0,1\}^n \)

\[
\Pr_{(k_0,\ldots,k_r)} \left[ \sum_i F_3^3(x_i) = 0^{c_n} \right] < 2^{-\Omega(n)}
\]

which will complete the proof because \( z \neq 0 \Rightarrow \Pr_{k_{r+1}} \left[ (z, k_{r+1}) = 0 \right] = 1/2 \).

For \( d = 2 \), this probability is 0 simply because \( x_1 \neq x_2 \Rightarrow F_3^3(x_1) \neq F_3^3(x_2) \) because each component of \( F_3 \) is injective.

Fix distinct \( x_1, x_2, x_3 \in \{0,1\}^n \). Fix any values for \( (k_0,\ldots,k_{r-2}) \), the round keys used prior to round \( r - 1 \). Let \( y_i \in \{0,1\}^{c_n-1} \) denote the state of the computation of \( F_3(x_i) \) immediately prior to the XOR with round key \( k_{r-1} \) in round \( r - 1 \), and let \( \Delta_i := y_i \oplus y_i \) denote the differences of the \( y_i \). Let \( z_1, z_2, z_3 \) be jointly-distributed random variables, over a uniform choice of \( k_{r-1} \), defined by \( z_i := y_i \oplus k_{r-1} \); note that \( (z_1, z_2, z_3) \) is uniformly distributed over all tuples with differences \( \Delta_i \).

Fix any \( j \leq m \), and let \( z_j \) denote the \( j \)th bundle of \( z_1 \) and \( \Delta_j \) denote the \( j \)th bundle of \( \Delta_i \). We wish to bound

\[
\Pr_{k_{r-1}} \left[ (z_1)^{2^b-2} + (z_1 + \Delta_2)^{2^b-2} + (z_1 + \Delta_3)^{2^b-2} = 0 \right] \quad (2.8)
\]

the probability that the outputs of the \( j \)th S-box in round \( r \) sum to 0. If \( \Delta_1 = \Delta_2 = 0 \), then the equation is satisfied iff \( z_1 = 0 \), in which case \( (2.8) = 2^{-b} \). Now assume that at least one of \( \Delta_1, \Delta_2 \) are not zero. If we assume that \( z_1 \notin \{0, \Delta_1, \Delta_2\} \), then we may multiply both sides of the equation by \( \prod_i (z_1 + \Delta_i) \neq 0 \) to get a quadratic polynomial in \( z_1 \). Thus, there are at most
Finally, because each bundle of $k_{r-1}$ is chosen independently, and because the remaining steps in round $r$ are linear, we have

$$\Pr_{(k_0,\ldots,k_r)} \left[ \sum_{i \leq 3} F_3^*(x_i) = 0^{c^r n} \right] < \left( \frac{6}{2^b} \right)^{c^r - 1 m} = 2^{-\Omega(n)}. \quad \square$$

Note that this proof does not use any properties of the code $C_\epsilon$ aside from injectivity. We remark why this proof does not show that $F_3$ is almost $d$-wise independent for $d \geq 4$. When the number of inputs $d$ is even, the equation in (2.8) is satisfied for all values of $z_1$ iff $\Delta_1, \ldots, \Delta_d$ can be partitioned into $d/2$ pairs such that the two values in each pair are equal. Indeed, for even $d \in (2, 2^b]$ it is possible to construct a set $\{\Delta_1, \ldots, \Delta_d\}$ which admits such a partition for all $j$ and yet satisfies the minimum-distance property of $C_\epsilon$ (which guarantees that $\geq \delta c^{r-1} m$ bundles of each $\Delta_i$ are non-zero for $i > 1$, and further that $\geq \delta c^{r-1} m$ bundles of $(\Delta_i \oplus \Delta_j)$ are non-zero for all $i \neq j$). However, it seems counterintuitive that the differences at round $r$ would satisfy such a specialized property with noticeable probability, and we believe that this proof can be extended to higher values of $d$.

### 2.3.4 Candidate 4

For our next candidate, we choose the extreme setting of $b = n$ and $r = 1$, which means that the function is computed over one round and essentially consists of just a single S-box. More specifically, the function is indexed by a seed $(k_0, k_1) \in \{0, 1\}^{2n}$, and is computed as

$$F_4(x) := (x + k_0)^{2^{n-2}} + k_1.$$

Though $F_4$ does indeed preserve resistance to differential and linear cryptanalysis, we note that the seed can be recovered with four known plaintext/ciphertext pairs, using an attack similar in spirit to the so-called interpolation attack of [JK01].

**Claim 2.** Let $F_4$ be the above function indexed by $k_0, k_1 \in \{0, 1\}^n$. Let $\{(p_i, c_i)\}_{1 \leq i \leq 4}$ be any set such that $c_i = F_4(p_i)$ for all $i$ and $p_i \neq p_j$ for $i \neq j$. Then, with probability $(1 - 1/2^{n-2})$ over $k_0$, the values of $k_0$ and $k_1$ can be recovered from $\{(p_i, c_i)\}_i$.

**Proof.** The attack is performed by using the four pairs to create two equations over $\text{GF}(2^n)$ that are linear in the seed, as follows. Assume that $k_0 \notin \{p_i\}_i$, which happens with probability
(1 − 1/2^{n−2}). Then the equation

\[(c_i + k_1) \cdot (p_i + k_0) = 1\]

holds for 1 ≤ i ≤ 4. We can rewrite these equations as

\[k_0k_1 + c_i k_0 + p_i k_1 + c_i p_i = 1. \quad (2.9)\]

If we sum (2.9) for i = 1, 2, the quadratic terms cancel and we obtain

\[(c_1 + c_2)k_0 + (p_1 + p_2)k_1 + (c_1p_1 + c_2p_2) = 0.\]

Summing (2.9) for i = 3, 4 gives another linear equation in \(k_0, k_1\). The attack concludes by solving the two linear equations.

The function \(F_4\) can be seen as a concrete instantiation of the Even-Mansour cipher [EM97] where the random permutation is replaced with (the asymptotic version of) the AES S-box. This cipher is easily breakable as we have just observed, but we now consider a slight modification to \(F_4\) that is not susceptible to this simple attack, and furthermore fools all parity tests that look at \(\leq 2^{0.9n}\) outputs. The modified function \(F'_4 : \{0, 1\}^n \rightarrow \{0, 1\}\) is defined as follows:

\[F'_4(x) := \langle (x + k_0)^{2^n−2}, k_1 \rangle.\]

In other words, we combine the AES S-box with the Goldreich-Levin hardcore predicate [GL89]. Note that we now output only a single bit. This modification – replacing the second XOR with an inner product – can also be applied to the Even-Mansour cipher. We consider it an interesting question to what extent the assumptions necessary for the pseudorandomness of Even-Mansour can be relaxed in this setting. (In their setting, the assumption is that all parties have oracle access to a truly random permutation.)

The next theorem shows that \(F'_4\) fools all parity tests that look at \(\leq 2^{0.9n}\) outputs. This result is reminiscent of the “exponentiation” small-bias generator in [AGHP92], where the \(x\)-th output bit is \(\langle k_0^x, k_1 \rangle\). Indeed, our proof is inspired by theirs. However we face the extra difficulty that the polynomials we work with are not of low degree.

**Theorem 2.2.5.** For any choice of \(d \leq 2^n\), \(F'_4\) is a \(d\)-wise small-bias generator with error \(d/2^n\).

That is, for any distinct \(a_1, \ldots, a_d \in \{0, 1\}^n\):

\[\left| \Pr_{k_0, k_1} \left[ \sum_{i=1}^{d} F'_4(a_i) = 0 \right] - \frac{1}{2} \right| < \frac{d}{2^n}.\]
Proof. Fix any distinct choices of \( a_1, \ldots, a_d \). Then, identifying elements of \( GF(2^n) \) with elements of \( \{0,1\}^n \), we have
\[
\sum_{i \leq d} \mathcal{F}_i'(a_i) = \sum_{i \leq d} \langle (a_i + k_0)^{2^n-2}, k_1 \rangle = \left\langle p(x) := \sum_{i \leq d} (a_i + k_0)^{2^n-2}, k_1 \right\rangle.
\]
We now show that the polynomial \( p(x) = \sum_{i \leq d} (a_i + x)^{2^n-2} \) has at most \( 2d-1 \) distinct roots. This will conclude the proof because when \( k_0 \) is not a root of \( p(x) \), we have \( \Pr_{k_1} [\langle p(k_0), k_1 \rangle = 0] = 1/2 \). Therefore,
\[
\left| \Pr_{k_0,k_1} \left[ \sum_{i=1}^{d} \mathcal{F}_i'(a_i) = 0 \right] - \frac{1}{2} \right| \leq \frac{1}{2} \Pr_{k_0} [p(k_0) = 0] \leq \frac{d}{2^n}.
\]
To show the bound on the number of roots, define the following polynomials:
\[
\overline{p}(x) := p(x) \cdot \prod_{i \leq d} (a_i + x) = \sum_{i \leq d} \left( (a_i + x)^{2^n-1} \prod_{j \neq i} (a_j + x) \right),
\]
\[
\overline{p}_*(x) := \prod_{i \leq d, j \neq i} (a_j + x).
\]
Observe that any root \( y \) of \( p(x) \) is also a root of \( \overline{p}(x) \). Moreover, note for any \( y \not\in \{a_j : j \leq d\} \), \( \overline{p}(y) = \overline{p}_*(y) \), using the identity \( y^{2^n-2} = 1 \) valid for any \( y \neq 0 \).

Also observe that \( \overline{p}_*(x) \) is not identically zero. Indeed, by inspection, the constant term of the polynomial \( \overline{p}_*(x + a_1) \) is \( \prod_{j \neq 1} (a_j + a_1) \), which is non-zero because the \( a_j \) are distinct; therefore \( \overline{p}_*(x + a_1) \) is not identically zero, and so neither is \( \overline{p}_*(x) \). Since \( \overline{p}_*(x) \) is a non-zero polynomial of degree \( d-1 \), it has at most \( d-1 \) distinct roots.

So, if \( p(x) \) has \( r \) roots, also \( \overline{p} \) has \( r \) roots. At least \( r - d \) of these do not belong to \( \{a_j : j \leq d\} \), and so they are also roots of \( \overline{p}_*(x) \). Therefore, \( r - d \leq d - 1 \), or \( r \leq 2d - 1 \).

By Braverman’s result [Bra09] (cf. [Baz09, Raz09]), we obtain that \( \mathcal{F}_i' \) also fools small-depth \( \text{AC}^0 \) circuits of any size \( w = 2^{o(1)} \) (that look at only \( w \) fixed output bits of the candidate).

Indeed, fix any function \( w = 2^{o(1)} \) and any constant \( d = O(1) \); let \( N := 2^n \). By Theorem 2.2.5, any \( w \) output bits have bias \( < w/N \). By [AGM03], for any \( k \leq w \), the output distribution on those \( w \) bits is \( w^k w/N \)-close to a \( k \)-wise independent distribution. By [Bra09], \( k = \lg^{O(d^2)} w \leq n^{o(1)} \) is sufficient to fool circuits of depth \( d \) with error \( 1/w \). Hence the overall error will be \( 1/w = 1/2^{o(1)} \) plus
\[
\frac{w^k w}{N} = \left( \frac{2^{o(1)}}{N} \right)^{n^{o(1)}} \leq \frac{1}{\sqrt{N}},
\]
for a total of \( 1/w + 1/\sqrt{N} = O(1/w) \).
CHAPTER 2. PSEUDORANDOM FUNCTIONS VIA SUBSTITUTION-PERMUTATION NETWORKS

Efficiency. As noted in §2.2, \( F'_4 \) is computable by Boolean circuits of size \( O(n \log^2 n \log \log n) \) and \( \text{TC}^0 \) circuits of size \( n^{O(1)} \).

2.3.5 Candidate 5

Our final candidate \( F_5 \) preserves the structure of AES almost exactly. For any \( n \) that is a multiple of 32, we set \( b = 8 \), \( m = n/8 \), and \( r = n \), and we use \( S(x) := x^{2^8-2} \). The linear transformation \( M \) is of a slightly different form than that of the previous candidates, which we explain now.

\( M \) is computed in two (linear) steps. In the first step, a permutation \( \pi : [m] \rightarrow [m] \) is used to shuffle the \( b \)-bit bundles of the state; namely, bundle \( i \) moves to position \( \pi(i) \). \( \pi \) is computed as follows. First, the \( m \) bundles are placed column-wise into a \( 4 \times m/4 \) matrix. Then row \( i \) of the matrix \((0 \leq i < 4)\) is shifted circularly to the left by \( i \) places, and finally the bundles are extracted column-wise from the new matrix.

In the second step, a maximal-branch-number matrix \( \phi \in \text{GF}(2^8)^{4 \times 4} \) is applied in parallel to each consecutive group of 4 bundles.

Efficiency: small circuits. In each round, the \( O(n) \) instances of \( S \) and \( \phi \) each perform computations on a constant number of bits; because permuting the bundles and adding the round key can also be done with \( O(n) \) wires, each round of \( F_5 \) can be computed by a circuit of depth \( d = O(1) \) and size \( w = O(n) \). Thus the entire \((r\text{-round})\) circuit for \( F_5 \) has depth \( d = O(n) \) and size \( w = O(n^2) \).

Efficiency: fast Turing machines. Similarly, for any fixed seed \( k \), each round of \( F_5 \) can be computed in time \( O(n) \) on a single-tape Turing machine with \( O(n^2) \) states. To do so, we encode the bundles on the tape so that the matrix used by \( \pi \) is written column-wise. As before, the \( O(n) \) instances of \( S \) and \( \phi \) in a single round can be done in time \( O(n) \). To see that \( \pi \) can also be computed in time \( O(n) \), note that due to the column-wise representation each bundle needs to move \( \leq 3 \) places away, except for the 6 bundles which are shifted circularly to the other end of the tape. Finally, encoding the \( O(n^2) \)-bit seed in the TM’s state transitions, the addition of each round key also takes time \( O(n) \). Therefore, the \( r = n \) rounds of \( F_5 \) can be computed in time \( O(n^2) \).

Alternatively, consider the Turing machine variant with two tapes, in which the first tape is read-only and contains the \( n \)-bit input followed by the \( n(n+1) \)-bit seed, the second tape is
read/write, and the TM has $O(1)$ states. Then $F_5$ can again be computed in time $O(n^2)$ exactly as described above, because in round $i$ only bits $in + 1, \ldots, in + n$ of the seed are used.

### 2.4 More related work

Hoory, Magen, Myers and Rackoff [HMMR05] and Brodsky and Hoory [BH08], building on work by Gowers [Gow96], study the random composition of a family of permutations. The SPN structure can be seen as falling into this framework, by taking each round as an individual permutation chosen randomly by the key. However, the permutations constructed in these works do not have the form of an SPN round, and furthermore the circuit complexity of the composed permutations is not of interest to them (their constructions have size and depth $\Omega(n^3)$).

**Block cipher modes.** We note that common methods of extending fixed-input-length block ciphers to domains of arbitrary size, the so-called “modes of operation”, are not sufficient to construct PRF starting from secure block ciphers. This is because while (some of) these modes preserve the security of the fixed-input-length block cipher, they do not provide hardness that grows super-polynomially with the input length which we explain now. For concreteness we focus here on the widely-used CBC-MAC mode (cf. [BKR00]), though similar attacks hold for all modes of which we are aware.

For a block cipher $f : \{0,1\}^\ell \to \{0,1\}^\ell$, the function $\text{CBC-MAC}_f : \{0,1\}^{n\ell} \to \{0,1\}^\ell$ is computed as follows on input $x = (x_1, \ldots, x_n) \in \{0,1\}^{n\ell}$: first set $y_0 := 0$, then compute $y_i := f(x_i \oplus y_{i-1})$ for $i = 1, \ldots, n$, and finally output $y_n$. The point now is that even if $f$ has maximum possible hardness $2^{\Omega(\ell)}$ as a PRF, and even if uniform and independent keys are used for each of the $n$ evaluations of $f$, $\text{CBC-MAC}_f$ has hardness $\leq 2^{O(\ell) \cdot n^{O(1)}}$ as a PRF. Briefly, the attack works as follows. First find two inputs $x \neq x'$ such that $x_n = x'_n = 0^\ell$ and $\text{CBC-MAC}_f(x) = \text{CBC-MAC}_f(x')$. This can be done in time $2^{O(\ell) \cdot n^{O(1)}}$ by the pigeonhole principle. Now for a string $w \in \{0,1\}^\ell$, let $x_w := (x_1, \ldots, x_{n-1}, w)$ and $x'_w := (x'_1, \ldots, x'_{n-1}, w)$. Then for any $w$ we have $\text{CBC-MAC}_f(x_w) = \text{CBC-MAC}_f(x'_w)$, while for a truly random function this holds only with probability $2^{-\ell}$.

**Natural proofs.** Finally we mention a way in which the efficiency of our constructions relates to techniques for proving circuit lower bounds. One of the oldest and most elusive goals of computational complexity is to prove strong lower bounds against efficient circuits, i.e. to give an explicit function that is not computable by such circuits. Researchers have been successful
in explaining the lack of progress in this area by pointing out several “barriers,” i.e. establishing that certain proof techniques will not give new lower bounds [BGS75, RR97, AW08].

Of particular interest to us is the Natural Proofs work by Razborov and Rudich [RR97]. They make the following two observations. First, most lower-bound proofs that a certain function $f : \{0,1\}^n \rightarrow \{0,1\}$ cannot be computed by circuits $C$ (e.g., $C =$ circuits of size $n^2$) entail an algorithm that runs in time polynomial in $N := 2^n$ and can distinguish truth-tables of $n$-bit functions $g \in C$ from truth-tables of random functions (i.e., a random string of length $N$). (For example, the algorithm corresponding to the restriction-based proof that Parity is not in $\text{AC}^0$ [Ajt83, FSS84], given $f : \{0,1\}^n \rightarrow \{0,1\}$, checks if there is one of the $2^{O(n)} = N^{O(1)}$ restrictions of the $n$ variables that makes $f$ constant.) Informally, any proof that entails such an algorithm is called “natural.”

The second observation is that, under standard hardness assumptions, no algorithm such as the above one exists when $C$ is a sufficiently rich class. This follows from the existence of PRFs with security $2^{s \Omega(1)}$ where $s$ is the seed length (e.g. [GGM86, HILL99, NR04, HRV10, VZ12, VZ13, BPR12]) by setting $s := n^c$ for a sufficiently large $c$.

The combination of the two observations is that no natural proof exists against circuits of size $n^c$, for some constant $c \geq 2$.

Moreover, the PRF constructions of [NR04, BPR12] are implementable in $\text{TC}^0$, pushing the above second observation “closer” to the frontier of known circuit lower bounds. For completeness we also mention that [NR04] achieves seed length $s = O(n^2)$ and is a candidate to having hardness $2^{\Omega(n)}$ under elliptic-curve conjectures.

However, the natural proofs barrier still has a significant gap with known lower bounds, due to the lack of sufficiently strong PRFs. For example, there is no explanation as to why one cannot prove superlinear-size circuit lower bounds. For this one would need a PRF $f_k : \{0,1\}^n \rightarrow \{0,1\}$ that is computable by linear-size circuits (hence in particular with $|k| = O(n)$) and with exponential hardness $2^n$. (So that, given $n$, if one had a distinguisher running in time $2^{O(n)}$, one could pick a PRF on inputs of length $bn$ for a large enough constant $b$, to obtain a contradiction.)

The work by Allender and Koucký [AK10] brings to the forefront another setting where the Natural Proofs barrier does not apply: proving lower bounds on $\text{TC}^0$ circuits of size $n^{1+\epsilon}$ and depth $d$, for any $\epsilon > 0$ and large enough $d = d(\epsilon)$. (As mentioned above, the Naor-Reingold PRF requires larger size.) This setting is especially interesting because [AK10] shows that such a lower bound for certain functions implies a “full-fledged” lower bound for $\text{TC}^0$ circuits of
polynomial-size. Moreover even if the first lower bound were natural, the latter would not be, thus circumventing the TC\(^0\) PRFs.

Another long-standing problem is to exhibit a candidate PRF in ACC\(^0\).

Of course, circuit models such as the above ones are only some of the models in which the gap between candidate PRF and lower bounds is disturbing. Other such models include various types of Turing machines, and small-space branching programs. For example, there is no explanation as to why the lower bounds for single-tape Turing machines stop at quadratic time, cf. [KN97, §12.2].

Assuming the exponential security of some of our candidates, our work narrows this gap in three ways. First, Candidates 2 and 4 are computable by quasilinear-size Boolean circuits. Second, Candidate 3 is computable by TC\(^0\) circuits of size \(n^{1+\epsilon}\) and depth \(d = d(\epsilon)\) for any \(\epsilon > 0\). Third, for each fixed seed \(k\) Candidate 5 is computable in time \(O(n^2)\) by a single-tape Turing machine with \(O(n^2)\) states (note that the fixed-seed setting suffices for the Natural Proofs connection).

### 2.5 Future directions

Two obvious directions for future work are to extend the analysis of \(\mathcal{F}_1\) to handle inverse queries (necessarily choosing the S-box as a random permutation), and to extend Theorem 2.2.4 to prove almost \(d\)-wise independence of \(\mathcal{F}_3\) for \(d > 3\). A more foundational question left unanswered is to understand how the degree of each output bit of an SPN (as a polynomial in the input bits) is affected by the degree of the S-box and by the “mixing” properties of the linear transformation.

Exploring other choices of the S-box besides inversion may lead to more efficient constructions, and utilizing other properties of the linear transformation besides maximal-branch-number may allow stronger proofs of security. This could potentially give a (plausibly secure) SPN computable by circuits of size \(O(n)\). Recall that a PRF computable with size \(O(n)\) and with security \(2^n\) would bring the Natural Proofs barrier to the current frontier of lower bounds against unbounded-depth circuits.

Abstracting from the SPN structure, one may arrive to the following paradigm for constructing PRF: alternate the application of (1) an error-correcting code and (2) a bundle-wise application of any local map that has high degree over GF(2) and resists attacks corresponding to linear and differential cryptanalysis. This viewpoint may lead to a PRF candidate computable in ACC\(^0\), since for (1) one just needs parity gates, while, say, taking parities of suitable mod 3
maps one should get a map that satisfies (2). However a good choice for this latter map is not clear to us at this moment.

We believe a good candidate PRF should be the simplest candidate that resists known attacks. As noted in [DR02], some of the choices in the design of AES are not motivated by any known attack, but are there as a safeguard (for example, one can reduce the number of rounds and still no attack is known). While this is comprehensible when having to choose a standard that is difficult to change or when deploying a system that is to be widely used, one can argue that a better way for the research community to proceed is to put forth the simplest candidate PRF, possibly break it, and iterate until hopefully converging to a secure PRF. We view our work as a step in this direction.
Chapter 3

Leakage-resilient circuits via iterated group products

This chapter is organized as follows. We first give an overview of our results in §3.1. In §3.2 we give our main construction of a leakage-resilient compiler. In §3.3 we prove lower bounds for computing iterated group products in a variety of computational models, establishing the security of our construction. In §3.4 we give some missing details from §3.2 and analyze our construction in the “only computation leaks” (OCL) model. In §3.5 we extend our construction to multiple queries. We conclude with some future directions in §3.6.

3.1 Overview

For simplicity, we first focus on the setting where the adversary makes a single query to the circuit, and we do not use any secure hardware. Defined next, our compiler is randomized and takes two inputs: a circuit $C : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$, and a value $k \in \{0,1\}^n$ for $C$’s second input. It outputs a circuit $\hat{C} : \{0,1\}^n \rightarrow \{0,1\}^n$ that is functionally equivalent to $C(\cdot, k)$ (we choose to omit a $k$ subscript though $\hat{C}$ depends on $k$). The only parts of $\hat{C}$ that depend on $k$ and the random coins are the values of its constant gates; the rest is determined by $C$. The adversary depends on $C$ and thus knows everything about $\hat{C}$ except the values of certain constant gates.

The adversary then selects both an input $x$ to the circuit and a leakage function to be evaluated on the wires of the circuit. The requirement that the adversary “learns nothing” from the output of the leakage function is formalized by providing an efficient simulator $S$. $S$ sees only the input $x$ and output $\hat{C}(x)$ of the circuit, as well as the circuit $C$ which is assumed to be public, and produces a set of wire values that is indistinguishable from the real set of wire values by the leakage function. Throughout the paper we will use $|C|$ to denote the number of wires in a circuit $C$, which is the input length of the leakage functions.

**Definition 3.1.1** (Leakage-secure compiler). Let $\text{Comp}(\cdot, \cdot)$ denote a randomized algorithm that takes as input a circuit $C : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$ and a string $k \in \{0,1\}^n$. For a set of functions $\mathcal{L}$, $\text{Comp}$ is an ($\mathcal{L}, \epsilon$)-leakage-secure compiler if the following properties hold.
1. (Structure.) For every $C$ and $k$, $\text{Comp}(C, k)$ outputs a circuit $\hat{C} : \{0,1\}^n \to \{0,1\}^n$ which is completely determined by $C$ except for the values of its constant gates.

2. (Correctness.) For every $C$ and $k$ and every $x \in \{0,1\}^n$, $\hat{C}(x) = C(x, k)$ with probability 1 over the choice of $\hat{C} \leftarrow \text{Comp}(C, k)$.

3. (Security.) There exists a randomized polynomial-time algorithm $S$ such that for every $C$ and $k$, every $x \in \{0,1\}^n$, and every $\ell \in L$ with domain $\{0,1\}^{\hat{C}}$:

$$\Delta(\ell(\hat{W}), \ell(S(C, x, \hat{C}(x)))) \leq \epsilon$$

where $\hat{W} \in \{0,1\}^{\hat{C}}$ denotes the values carried by the wires of $\hat{C}(x)$, and the statistical distance $\Delta$ is over the choice of $\hat{C} \leftarrow \text{Comp}(C, k)$ and the random coins of $S$.

We show how to efficiently compile any circuit $C$ into a leakage-resilient circuit $\hat{C}$ such that any function on the wires of $\hat{C}$ that leaks information during some computation $\hat{C}(x)$ yields advantage in computing iterated group products over the alternating group $A_u$, which recall is the group of even permutations of a set of size $u$. (For background on this group, see e.g. [Wil09, §2].) The security of our construction is proved against leakage classes $L$ for which iterated products over $A_u$ are hard in the following sense. As discussed later we exploit specific properties of $A_u$, but when possible we present things over any group $G$.

Definition 3.1.2 ($\epsilon$-fooled). Let $G$ be a group (whose operation is written multiplicatively). For $\alpha \in G$ and $t \in \mathbb{N}$, let $D_\alpha$ denote the uniform distribution over $\{(x_1, \ldots, x_t) \in G^t \mid \prod_i x_i = \alpha\}$, and let $U_{G^t}$ denote the uniform distribution over $G^t$.

Then a set of functions $L$ is $\epsilon$-fooled by $G^t$ if $\Delta(\ell(D_\alpha), \ell(U_{G^t})) \leq \epsilon$ for every $\alpha \in G$ and every $\ell \in L$ with domain $G^t$.

We will use the notation $D_\alpha, U_{G^t}$ throughout this chapter. Our security reductions are computable by simple, local (a.k.a. $\text{NC}^0$) functions.

Definition 3.1.3 (Local extension). A function $f : G^t \to G^*$ is a $d$-local function if each output element depends on at most $d$ input elements. If $d = O(1)$ as a function of $t$, we simply say local. For a set of functions $L$, the $d$-local extension of $L$ is the set of all functions $\ell(f(\cdot))$ where $\ell \in L$ and $f$ is a $d$-local function.

Our compiler is given by the following main theorem.
Theorem 3.1.4. For every polynomial-time computable function $t = t(n)$ and every sequence of groups $G = G(t)$ parameterized by $t$, there is a compiler $\text{Comp}$ for which the following holds.

1. For every $C : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$ and $k \in \{0,1\}^n$, $\text{Comp}(C, k)$ runs in time $\text{poly}(|C|, t, \log |G|)$ and outputs a circuit $\widehat{C}$ of size $O(|C| \cdot t^2 \cdot \log |G|)$.

2. For every set of functions $\mathcal{L}$ and every $\epsilon > 0$, if the 4-local extension of $\mathcal{L}$ is $\epsilon$-fooled by $G^t$ then $\text{Comp}$ is an $(\mathcal{L}, \epsilon \cdot t \cdot |C|)$-leakage-secure compiler.

Note that making $t$ smaller reduces the size overhead of $\widehat{C}$, but that larger values of $t$ are necessary to find rich classes $\mathcal{L}$ that are fooled by $G^t$.

To instantiate our construction we prove in §3.3 that $(A_u)^t$ fools a number of well-studied classes of functions (with parameters polynomially related to $t$). For most of these results we can and will choose $u = 5$. One class is that of number-on-forehead multiparty protocols introduced by Chandra, Furst, and Lipton [CFL83], which are formally defined in §3.3.1; here our result relies on the long-standing lower bound by Babai, Nisan, and Szegedy [BNS92], whose proof is increasingly streamlined in [CT93, Raz00, VW08]. Another is the class $\text{AC}^0$ of bounded-depth And/Or/Not circuits augmented with few gates computing arbitrary symmetric functions, such as parity and majority. This is the richest circuit class for which super-polynomial average-case lower bounds are known [Vio07]. In fact, one can allow few gates whose local extension has low number-on-forehead communication under any partition, such as polynomial threshold functions [Nis93, Vio]. We also consider the class $\text{TC}^0$ of bounded-depth circuits of majority gates; for this class no lower bound is known, and our results rely on the standard complexity assumption $\text{TC}^0 \neq \text{NC}^1$.

Obviously $(A_u)^t$ does not fool the class $\text{NC}^1$ of $O(\log t)$-depth, fan-in-2 circuits when $u = O(1)$, for such circuits can simply compute the product. However we show that when choosing $u = t$, $(A_u)^t$ fools $\text{NC}^1$ under the standard complexity assumption $\text{NC}^1 \neq \text{L}$. Our proof builds on the work of Cook and McKenzie [CM87] who show that computing products over the symmetric group $S_u$ is complete for $\text{L} = \text{logarithmic space}$.

The following theorem summarizes the results above. These results can also be seen as giving compression bounds, similar to the work of Dubrov and Ishai [DI06] (see also [HN10, Dru12] and others). In fact, we essentially recover for $A_5$-products the parameters of the [DI06] compression bound against $\text{AC}^0$ (building on their result), and also prove compression bounds against stronger classes. (In the following, $O_d(\cdot)$ and $\Omega_d(\cdot)$ hide constants that depend only on $d$.)
Theorem 3.1.5. \( (A_u)^t \) \( \epsilon \)-fools \( \mathcal{L} \) for:

1. \( \mathcal{L} = \) number-on-forehead protocols with \( s \) parties communicating and outputting \( \leq c \) bits, under a specific partition of the input; \( \epsilon = 2^{c-O(t/(s^2))} \); \( u = 5 \).

2. If \( \text{NC}^d \neq \text{L} \) then for every \( k \) and infinitely many \( t \), \( \mathcal{L} = \text{NC}^d \) circuits with size \( \leq t^k \) and \( k \log t \) bits of output; \( \epsilon = t^{-k} \); \( u = t \).

3. If \( \text{TC}^0 \neq \text{NC}^1 \) then for every \( k \) and infinitely many \( t \), \( \mathcal{L} = \text{TC}^0 \) circuits with size \( \leq t^k \) and \( k \log t \) bits of output; \( \epsilon = t^{-k} \); \( u = 5 \).

4. \( \mathcal{L} = \text{AC}^0 \) circuits with depth \( \leq d \), size \( \leq t^{O_d(\log t)} \), an additional \( O_d(\log^2 t) \) arbitrary symmetric gates, and \( t^{0.1} \) bits of output; \( \epsilon = t^{-\Omega_d(\log t)} \); \( u = 5 \).

5. \( \mathcal{L} = \text{AC}^0 \) circuits with depth \( \leq d \), size \( \leq 2^{O_d(t^{(1-\delta)/d})} \), and \( t^\delta \) bits of output, for any \( \delta < 1 \); \( \epsilon = 2^{-\Omega_d(t^{(1-\delta)/d})} \); \( u = 5 \).

The straightforward combination of Theorems 3.1.4 and 3.1.5 gives an \( (\mathcal{L}, \epsilon) \)-secure compiler for the circuit classes listed in items 2-5 of the latter, choosing \( t = |C| \) for the following corollary. The combination is less straightforward for protocols, which are not closed under composition with arbitrary local functions. We obtain item 1 of the following corollary by showing (in §3.4) that the local extension of a number-in-hand protocol is computable by a number-on-forehead protocol.

Corollary 3.1.6. There is a single efficient compiler \( \text{Comp} \), outputting a circuit \( \widehat{C} \) of size \( |\widehat{C}| = \text{poly}(|C|) \), that is an \( (\mathcal{L}, \epsilon) \)-leakage secure compiler for each of the following.

1. \( \mathcal{L} = \) number-in-hand protocols with \( s \) parties communicating and outputting \( \leq \delta \cdot |\widehat{C}|^{1/3} \) bits, for a fixed \( \delta > 0 \) and a fixed partition of \( \widehat{C} \) into \( s = O(1) \) sets; \( \epsilon = 2^{-\Omega(|\widehat{C}|^{1/3})} \).

2. If \( \text{NC}^d \neq \text{L} \) then for every \( k \) and infinitely many \( |C| \), \( \mathcal{L} = \text{NC}^d \) circuits with depth \( \leq k \log |\widehat{C}| \) and \( k \log |\widehat{C}| \) bits of output; \( \epsilon = |\widehat{C}|^{-k} \).

3. If \( \text{TC}^0 \neq \text{NC}^1 \) then for every \( k \) and infinitely many \( |C| \), \( \mathcal{L} = \text{TC}^0 \) circuits with size \( \leq |\widehat{C}|^k \) and \( k \log |\widehat{C}| \) bits of output; \( \epsilon = |\widehat{C}|^{-k} \).

4. \( \mathcal{L} = \text{AC}^0 \) circuits with depth \( \leq d \), size \( \leq |\widehat{C}|^{O_d(\log |\widehat{C}|)} \), an additional \( O_d(\log^2 |\widehat{C}|) \) arbitrary symmetric gates, and \( |\widehat{C}|^{0.01} \) bits of output; \( \epsilon = |\widehat{C}|^{-\Omega_d(\log |\widehat{C}|)} \).
5. $\mathcal{L} = \text{AC}^0$ circuits with depth \( \leq d \), size \( \leq 2^{O_d(|\hat{C}|^{(1-\delta)/3d})} \), and \( |\hat{C}|^{\delta/3} \) bits of output, for any \( \delta < 1 \); \( \epsilon = 2^{-\Omega_d(|\hat{C}|^{(1-\delta)/3d})} \).

In particular, our construction unconditionally resists leakage from functions such as parity, majority, inner product, and polynomial thresholds. Besides being well-studied, these functions break most previous constructions. For example, inner product breaks [DF12, GR12], and parity breaks [ISW03, FRR+10, Rot12]. Also, small TC$^0$ circuits can be shown to break at least one instantiation of the construction [JV10] using the fact that such circuits may compute division, cf. [All01]. In fact, we are only aware of one construction that is not easily broken in TC$^0$ (and thus also in NC$^1$). This is the construction [GR10] which relies on the Decisional-Diffie-Hellman assumption. It is broken by any leakage function that can decrypt a certain public-key cryptosystem based on it, but decryption here involves modular exponentiation (to a poly-length exponent); whether this is doable in small depth is an open problem. Finally, note that the last item shows that we recover the security of [FRR+10] against AC$^0$ leakage functions.

### 3.1.1 Multiple queries

We also consider the setting in which the adversary makes multiple, adaptive queries to the circuit \( \hat{C} \). As in the previous setting, each query consists of both an input to the circuit and a leakage function. The adversary is assumed to be computationally unbounded, except for the restriction on the leakage functions. We defer until §3.5 the formal definition of security in this setting.

If the number of queries \( q \) is fixed in advance and known to the compiler, then our construction in Theorem 3.1.4 can be extended with little difficulty to this setting. The resulting construction increases the size of \( \hat{C} \) by a factor of \( O(q) \) and likewise the security degrades by a factor of \( q \) (details omitted).

When the number of queries \( q \) is not a priori bounded, we adopt the approach of [FRR+10] and augment \( \hat{C} \) with a so-called secure hardware component. In our construction, this component is a randomized, inputless gate that on each execution outputs a sample from \( D_{\text{id}} \), where \( \text{id} \) denotes the identity element of \( A_5 \). We refer to such gates as \( D_{\text{id}} \)-gates, and any circuit that contains one as a \( D_{\text{id}} \)-circuit. The complexity of this component is comparable to the one in [FRR+10] which outputs a uniform bit vector with parity 0. (Secure hardware components are also used in [GR10, JV10], but there the components are not inputless and furthermore the distribution sampled is significantly more complex.)
To prove security in this setting, a slightly stronger property is required of \((A_5)\)' than what is given by Theorem 3.1.5. Specifically we require that, for every \(\ell \in \mathcal{L}\) and every \(\ell' \in \mathcal{L}\) that is chosen adaptively based on the output of \(\ell\), the distribution \((\ell(x), \ell'(x))\) when \(x \leftarrow D_\alpha\) has statistical distance \(\leq \epsilon\) from the corresponding distribution when \(x \leftarrow U_{G^t}\). We show in §3.5.1 that each of the classes \(\mathcal{L}\) listed in Theorem 3.1.5 has this property; the only difference is that for \(\text{AC}^0\) circuits with symmetric gates, we restrict the output length to \(O(\log^2 t)\).

We defer the details for now and simply state our result for multiple queries.

**Corollary 3.1.7.** There is a single efficient compiler \(\text{Comp}\), outputting a \(D_{\alpha}\)-circuit \(\hat{C}\) of size \(|\hat{C}| = O(|C|^3)\), that is a \(q\)-query (\(\mathcal{L}, \epsilon\))-leakage secure compiler for any \(q\) and each of the following.

1. \(\mathcal{L} = \text{number-in-hand protocols with } s \text{ parties communicating } \leq \delta \cdot |\hat{C}|^{1/3}\) bits, for a fixed \(\delta > 0\) and a fixed partition of \(\hat{C}\) into \(s = O(1)\) sets; \(\epsilon = q \cdot 2^{-\Omega(|\hat{C}|^{1/3})}\).

2. If \(\text{NC}^1 \neq \text{L}\) then for every \(k\) and infinitely many \(|C|\), \(\mathcal{L} = \text{NC}^1\) circuits with depth \(\leq k \log |\hat{C}|\) and \(k \log |\hat{C}|\) bits of output; \(\epsilon = q \cdot |\hat{C}|^{-k}\).

3. If \(\text{TC}^0 \neq \text{NC}^1\) then for every \(k\) and infinitely many \(|C|\), \(\mathcal{L} = \text{TC}^0\) circuits with size \(|\hat{C}|^k\) and \(k \log |\hat{C}|\) bits of output; \(\epsilon = q \cdot |\hat{C}|^{-k}\).

4. \(\mathcal{L} = \text{AC}^0\) circuits with depth \(\leq d\), size \(|\hat{C}|^{O_d(\log |\hat{C}|)}\), an additional \(O_d(\log^2 |\hat{C}|)\) arbitrary symmetric gates, and \(O_d(\log^2 |\hat{C}|)\) bits of output; \(\epsilon = q \cdot |\hat{C}|^{-\Omega_d(\log |\hat{C}|)}\).

5. \(\mathcal{L} = \text{AC}^0\) circuits with depth \(\leq d\), size \(2^{O_d(|\hat{C}|^{(1-\delta)/3d})}\), and \(|\hat{C}|^{\delta/3}\) bits of output, for any \(\delta < 1\); \(\epsilon = q \cdot 2^{-\Omega_d(|\hat{C}|^{(1-\delta)/3d})}\).

### 3.1.2 Only computation leaks

Finally we discuss the security of our construction in the “only computation leaks” (OCL) model due to Micali and Reyzin [MR04]. In this model, the compiler specifies a topologically-ordered partition \(P = (P_1, \ldots, P_r)\) on the wires of \(\hat{C}\). (Topologically-ordered means that for each \(P_i \in P\) and each wire \(w \in P_i\), \(w\)'s value in \(\hat{C}\)'s computation depends only on wires \(w'\) such that \(w' \in P_j\) for some \(j \leq i\).) Then for each \(P_i \in P\), the adversary chooses a leakage function to be applied to \(P_i\)'s wires during the computation \(\hat{C}(x)\) for its chosen input \(x\).

The goal in this setting is to tolerate leakage functions that are chosen adaptively for each \(P_i\) and are computationally unrestricted other than a bound on their output length. The work of Goldwasser and Rothblum [GR12] achieves this with a partition into \(O(|C|)\) sets each of size
$O(t^\omega)$ and tolerates $\Omega(t)$ bits of leakage per set, where $\omega$ is the matrix multiplication exponent. Furthermore, their construction does not use secure hardware components.

Without the topological ordering requirement, item 1 of Corollary 3.1.6 gives a partition into $O(1)$ sets each of size $O(|C| \cdot t^2 \cdot \log |G|)$ and tolerates $\Omega(t)$ bits of leakage per set. Next we refine this into a topologically-ordered partition with $O(t \cdot |C|)$ sets each of size $O(t \cdot \log |G|)$. The amount of leakage tolerated depends on the strength of communication lower bounds for number-on-forehead (NOF) protocols, as stated in the following theorem. For $s$-party NOF protocols, we refer to the canonical partition of $x \in G^t$ as the one in which player $i$’s forehead contains $x_i, x_{i+s}, \ldots, x_{i+t-s}$.

**Theorem 3.1.8.** Assume that 8-party NOF protocols communicating $\leq c$ bits are $\epsilon$-fooled by $G^t$ under the canonical partition of $x \in G^t$.

Then for each $C$ and $k$ there is a topologically-ordered partition $P$ on $\hat{C} := \text{Comp}(C, k)$ containing $O(t \cdot |C|)$ sets each of size $O(t \cdot \log |G|)$, such that $\text{Comp}$ from Theorem 3.1.4 is an $(\mathcal{L}, \epsilon \cdot t \cdot |C|)$-leakage secure compiler for $\mathcal{L} = \text{all OCL leakage functions that output } \leq \delta \cdot c/t \text{ bits per set in } P$, where $\delta$ is a constant that depends only on the maximum fanout of $C$.

Item 1 of Theorem 3.1.5 gives a lower bound for NOF protocols that achieves $c = \Omega(t)$ and $\epsilon = 2^{-\Omega(t)}$ for any constant number of parties. When the group size is constant, this is optimal up to constant factors as the entire input to the NOF protocol has size $t \cdot |G| = O(t)$. Unfortunately, plugging this into Theorem 3.1.8 does not give anything because $c/t < 1$.

In order to use Theorem 3.1.8 to achieve security in the OCL model, one must prove communication lower bounds that grow with the size of the group. For example one could ask whether, for some $\delta > 0$, NOF protocols communicating $c = \delta t \log t$ bits are fooled by $(A_t)^t$. By Theorem 3.1.8, this would give a partition of $\hat{C}$ into $O(t \cdot |C|)$ sets of size $O(t^2 \log t)$ and allow $\Omega(\log t)$ bits of leakage per set. By comparison, recall that [GR12] achieves a partition into $O(|C|)$ sets of size $O(t^\omega)$ and allows $\Omega(t)$ bits of leakage per set.

Interestingly, for groups such as $A_t$ whose elements are represented as permutations on a set $[t]$, one can only hope to prove communication lower bounds of the form $c = \Omega(t \log t)$. This is because a protocol that “traces” some point in $[t]$ through the permutations can distinguish any two fixed products with communication $O(t \log t)$. If one could instead obtain an $\Omega(t^2)$ lower bound against NOF protocols computing products over some group $G$ of size $|G| = 2^{\Theta(t)}$, this would give a compiler tolerating $\Omega(t)$ bits of leakage per set of size $O(t^2)$, essentially matching [GR12] (modulo the use of secure hardware). Because any group $G$ can be embedded in $S_{|G|}$ by
Cayley’s theorem, there is always a “tracing” protocol that communicates $O(t \log |G|) = O(t^2)$ bits. (In fact the trivial protocol that communicates its entire input already achieves this bound.) Thus we are posing the question of whether there is any group $G$ for which this is tight.

### 3.2 The construction

In this section we describe our main construction. Our compiler uses the general framework of the works [ISW03, FRR+10]. In this framework, to every wire of $C$ there corresponds in the compiled circuit $\hat{C}$ a “bundle” of wires which encode the same information. (In [ISW03, FRR+10] a bit $b$ is encoded by a bundle $x$ whose parity is $b$.) One then uses appropriate gadgets to simulate the computation of $C$ on the bundles. Note the distinction between gates and gadgets in $\hat{C}$: gadgets operate on bundles of wires, and are composed of gates that operate on individual wires.

The main differences between our construction and the ones in [ISW03, FRR+10] are in the encoding and in the gadgets. A side-benefit of our gadgets is that they allow for a more modular construction yielding an arguably more intuitive proof of security. Next we describe our encoding, our gadgets, and the proof of security. But first we make some remarks on the group used throughout.

**The choice of the group.** We exploit 3 properties of the alternating group $A_u$.

1. It fools various classes in the sense of Definition 3.1.2 (see Theorem 3.1.5).
2. It supports Barrington’s encoding of $\text{NC}^1$ computation [Bar89], which we use in the construction of the NAND gadget below. (This is implied by the group being non-solvable, which in turn is implied by each element being a commutator.)
3. It has specific elements that support a more efficient encoding of certain functions such as parity, improving on (ii). This is used in Theorem 3.1.5 to obtain improved parameters and in particular to match the parameters of the previous compression bound in [DI06].

We point out that (i) is not implied by (ii). Indeed, for (ii) the group $S_5$ is typically chosen. However $(S_5)^t$ does not even (1/2)-fool the 1-local extension of parity, which can compute the sign of the product permutation. This is because the sign of $D_\alpha$ always equals the sign of $\alpha$, whereas the sign of $U_{(S_5)^t}$ is equidistributed over $\{-1, 1\}$.

**The group encoding.** We encode a bit $b \in \{0, 1\}$ by a tuple of elements over a group $G$ as follows. Let $\text{id}$ denote $G$’s identity element, and fix an element $\text{id} \neq \alpha \in G$. Then
(x_1, \ldots, x_t) \in G^t$ encodes $b$ when

$$\prod_i x_i = \begin{cases} 
\text{id} & \text{if } b = 0 \\
\alpha & \text{if } b = 1.
\end{cases}$$

As in [Bar89], we can use any $\alpha$ for which there exists an element $\beta \in G$ such that $\alpha, \beta,$ and $\alpha \beta \alpha^{-1} \beta^{-1}$ are in the same conjugacy class. Equivalently, there must exist three elements $\beta, \gamma, \rho \in G$ such that the following two equations hold.

$$\gamma \alpha \gamma^{-1} = \beta \quad \rho \alpha \alpha^{-1} \beta^{-1} \rho^{-1} = \alpha. \quad (3.1)$$

For $G = A_5$ and using cycle notation, these values can be set as follows: $\alpha = (12345)$, $\beta = \rho = (14235)$, $\gamma = (12354)$.

For convenience we present the construction over $G$ as opposed to $\{0, 1\}$ and using gates for group multiplication and inversion. It is straightforward to obtain a construction over $\{0, 1\}$ and any standard basis by implementing group operations via bit operations. For example when $G = A_u$, we represent a permutation $\sigma$ using $(u \log u)$-bit pointwise representation as $(\sigma(1), \ldots, \sigma(u))$. Inversion and pairwise multiplication in can then be implemented by fan-in-2 circuits of depth $O(\log u)$, because each essentially amounts to indexing an element from an array of length $u$.

The **nand gadget.** We assume without loss of generality that $C$, the circuit input to the compiler, contains only fan-in-2 gates that compute the Nand function. We now describe the **nand gadget** that simulates each Nand gate in $C$. Given as input two bundles $x, y \in G^t$ with products in $\{\text{id}, \alpha\}$, the **nand gadget** outputs a bundle $z \in G^t$ that encodes the Nand of $x$ and $y$, i.e., that satisfies:

$$\prod_i z_i = \begin{cases} 
\text{id} & \text{if } \prod_i x_i = \prod_i y_i = \alpha \\
\alpha & \text{otherwise.}
\end{cases} \quad (3.2)$$

The output bundle $z \in G^t$ is computed by the following steps.

1. Set $y \leftarrow (\gamma \cdot y_1, y_2, \ldots, y_{t-1}, y_t \cdot \gamma^{-1})$. (This gives $\prod_i y_i \in \{\text{id}, \beta\}$.)

2. Compute $x^{-1} := (x_t^{-1}, \ldots, x_1^{-1})$ and similarly $y^{-1}$.

3. Compute $z \in G^{4t}$ by concatenating $(x, y, x^{-1}, y^{-1})$. (This gives $\prod_i z_i \in \{\text{id}, \alpha \beta \alpha^{-1} \beta^{-1}\}$.)

4. Set $z \leftarrow (\rho \cdot z_1, z_2, \ldots, z_{4t-1}, z_{4t} \cdot \rho^{-1})$. (This gives $\prod_i z_i \in \{\text{id}, \alpha\}$.)
5. Set \( z \leftarrow (z_{4t}^{-1}, \ldots, z_1^{-1} \cdot \alpha) \). (This maintains \( \prod_i z_i \in \{\text{id}, \alpha\} \) but if the product in step 4 was \( \alpha \) it is now \( \text{id} \), and vice versa.)

6. Compute and output \( z \in G^t \) by multiplying consecutive groups of 4 elements in \( z \):

\[
z := \left( \prod_{i=1}^{4} z_i, \prod_{i=5}^{8} z_i, \ldots, \prod_{i=4t-3}^{4t-1} z_i \right).
\]

From the equations (3.1) it can be verified that (3.2) is satisfied.

**Warm-up for the random gadget.** The second and last gadget that we need is called random and is essentially applied to every bundle in \( \hat{C} \) that corresponds to a wire in \( C \). This gadget has to satisfy two properties. First we need that on input \( x \in G^t \), the random gadget outputs a bundle \( z \in G^t \) that is distributed uniformly over \( \{z \in G^t \mid \prod_i z_i = \prod_i x_i\} \). This is necessary both for the correctness and security of the construction. The second property, necessary only for the security, is that given an input-output pair \((x, z)\) for this gadget, we should be able to compute locally a distribution on the gadget’s wires that is indistinguishable from the real distribution. (This allows us to replace the real distribution on the wires of \( \hat{C} \) with the one in which each random gadget is reconstructed. Then we can replace each bundle of wires in \( \hat{C} \) with a uniform bundle, which the simulator can do by itself, and blame any inconsistency on the reconstructor.) This property is called **local reconstructibility** and is a variant of the one in [FRR+10].

Before describing our gadget, we note that there is a simple gadget that satisfies the first property but not the second. Namely, choose \( r_1, \ldots, r_{t-1} \in G \) uniformly at random, and output

\[
z = (x_1 \cdot r_1, r_1^{-1} \cdot x_2 \cdot r_2, \ldots, r_{t-1}^{-1} \cdot x_t).
\]

(3.3)

Indeed, this basic re-randomization technique has been used to great effect in a number of works, e.g. [Kil88, FKN94, AIK06, GGH+08, AAW10]. However, this simple gadget does not satisfy local reconstructibility. One reason is that given \( x, z \), one can come up with values for the \( r_i \) that are consistent with each gate in the circuit if and only if \( \prod x_i = \prod z_i \). However, the latter is an \( \text{NC}^1 \)-hard question, whereas consistency of the \( r \) may be checked by, say, a DNF.

By contrast, one feature of our gadget is that given \( x, z \) one can produce consistent values for the wires even if \( \prod x_i \neq \prod z_i \). The catch is that in the latter case the values of certain constant gates are not chosen as in the correct implementation, but the leakage functions will not be able to distinguish this.
The random gadget. We now describe our gadget. The computation corresponds to replacing each pair \((r_i, r_i^{-1})\) in (3.3) with a pair \((R, L) \in G^t \times G^t\) such that \((\prod_j R_j) \cdot (\prod_j L_j) = \text{id}\), and then computing the multiplications in a specific order.

First, choose \(R^{(1)}, \ldots, R^{(t-1)} \in G^t\) uniformly at random. Next, choose \(L^{(2)}, \ldots, L^{(t)} \in G^t\) at random conditioned on
\[
\prod_j L^{(i)}_j = \left(\prod_j R^{(i-1)}_j\right)^{-1},
\]
for \(1 < i \leq t\). In the single-query setting, we think of Comp choosing these values and hardwiring them into \(\hat{C}\); in the multi-query setting, each pair \((R^{(i-1)}, L^{(i)})\) will be output by a secure hardware component. We will drop the superscripts on \(R\) and \(L\) when they are clear from context. Condition (3.4) implies the following equation.
\[
\left(\prod_j x \cdot \prod_j R^{(1)}_j\right) \cdot \left(\prod_j L^{(2)}_j \cdot x_2 \cdot \prod_j R^{(2)}_j\right) \cdots \left(\prod_j L^{(t)}_j \cdot x_t\right) = \prod_j x_i.
\]
So, we compute \(z\) by letting \(z_i \in G\) be the result of the \(i\)th parenthesized expression in (3.5). Clearly this \(z\) has the correct distribution. We perform each iterated multiplication by a depth-\(O(t)\) tree of fan-in-2 multiplication gates in a specific way, described now.

For \(z_1\), the product is computed in the straightforward way from left to right by a depth-\(t\) tree that computes each prefix product
\[
\lambda_m := x_1 \cdot \prod_{j=1}^m R_j = \lambda_{m-1} \cdot R_m
\]
for \(m = 1, \ldots, t\) in order, and outputs \(z_1 := \lambda_t\). The product for \(z_i\) is computed in the straightforward way from left to right as well.

Now let \(1 < i < t\). The product for \(z_i\) is computed by a depth-\(2t\) tree that multiplies “from the inside out”. That is, it computes in order a sequence \(\lambda_1, \ldots, \lambda_{2t-1}\) defined by \(\lambda_1 := L_t \cdot x_i\) and recursively for \(j = 1, \ldots, t - 1\) by
\[
\lambda_{2j} := \lambda_{2j-1} \cdot R_j,
\]
\[
\lambda_{2j+1} := L_{t-j} \cdot \lambda_{2j}
\]
and then outputs \(z_i := \lambda_{2t-1} \cdot R_t\).
By way of illustration, when $t = 3$ the sequence is computed as follows.

$$
\begin{align*}
\lambda_1 &= L_3 x_i \\
\lambda_2 &= L_3 x_i R_1 \\
\lambda_3 &= L_2 L_3 x_i R_1 \\
\lambda_4 &= L_2 L_3 x_i R_1 R_2 \\
\lambda_5 &= L_1 L_2 L_3 x_i R_1 R_2 \\
z_i &= \lambda_6 = L_1 L_2 L_3 x_i R_1 R_2 R_3.
\end{align*}
$$

It is interesting to note that that the size overhead of $O(t^2)$ in our construction comes from the fact that the random gadget has this size, and not from the NAND gadget which has size $O(t)$. This is in contrast to previous constructions using the parity encoding (e.g. [ISW03, FRR10]), for which it is not known how to compute Nand with this size. Improving the $O(t^2)$ overhead in any of these constructions, including ours, is an interesting open problem.

The following key lemma in this work shows that the random gadget is locally reconstructible. We say that $x, z \in G^t$ are plausible if it is possible for $\text{RANDOM}(x)$ to output $z$, i.e. if $\prod_i x_i = \prod_i z_i$.

**Lemma 3.2.1.** There is a poly($t$)-time computable distribution on 1-local functions $R_{\text{RANDOM}} : G^t \times G^t \rightarrow G^{[\text{RANDOM}]}$ for which the following holds. Let $W_{x \rightarrow z}$ denote the distribution on the wires of $z = \text{RANDOM}(x)$. For any $\ell$ with domain $G^{[\text{RANDOM}]}$ and any plausible $x, z \in G^t$, if

$$
\Delta(\ell(R_{\text{RANDOM}}(x, z)), \ell(W_{x \rightarrow z})) > \epsilon \cdot (t - 1)
$$

then some 1-local extension of $\ell$ is not $\epsilon$-fooled by $G$, i.e., there is a $g \in G$ and a 1-local function $f : G^t \rightarrow G^{[\text{RANDOM}]}$ such that

$$
\Delta(\ell(f(D_g)), \ell(f(U_{G^t}))) > \epsilon.
$$

**Proof.** We first describe an alternate procedure for generating $W_{x \rightarrow z}$. Fix any plausible $x, z \in G^t$. For the tree computing $z_1$, choose each $\lambda_j$ uniformly at random for $j = 1, \ldots, t - 1$, and compute each $R_j^{(1)} := \lambda_{j-1}^{-1} \cdot \lambda_j$ for $j = 1, \ldots, t$, defining $\lambda_t := z_1$ and $\lambda_0 := x_1$. Choose the wires for the tree computing $z_t$ analogously. Then for $i = 2, \ldots, t - 1$ in order, choose the wires for the tree computing $z_i$ as follows.

1. Choose $L^{(i)} \in G^t$ uniformly over vectors with product $= (\prod_j R_j^{(i-1)})^{-1}$. 


2. For \( j = 1, \ldots, t - 1 \) choose \( \lambda_{2j} \in G \) uniformly at random.

3. For \( j = 0, \ldots, t - 1 \) compute \( \lambda_{2j+1} := L^{(i)}_{i-j} \cdot \lambda_{2j} \), where \( \lambda_0 := x_i \).

4. For \( j = 1, \ldots, t \) compute \( R^{(i)}_j := \lambda_{2j-1}^{-1} \cdot \lambda_{2j} \), where \( \lambda_{2t} := z_i \).

To show that this distribution is identical to \( W_{x \to z} \), it is enough to observe that the vectors \( R^{(i)}, L^{(i)} \) are distributed correctly, i.e. uniformly conditioned on (3.4) and on the \( i \)th parenthesized expression in (3.5) equalling \( z_i \) for all \( i \). This is because the above procedure computes consistent wire values, and the \( R^{(i)}, L^{(i)} \) (along with \( x, z \)) determine the values of all other wires. \( R^{(1)} \) and \( L^{(t)} \) are clearly distributed correctly. Then for each \( 1 < i < t \), \( L^{(i)} \) is distributed correctly assuming that \( R^{(i-1)} \) is, \( R^{(i)}_j \) is uniform for \( 1 \leq j < t \), and \( R^{(i)}_t \) takes the unique consistent value.

This computation is sequential, due to the selection of \( L^{(i)} \) based on \( R^{(i-1)} \) in step 1. This selection is there to ensure condition (3.4). However by dropping this condition, we can break the dependencies between multiplication trees and give a local reconstructor. Namely, we define \( \mathcal{R}_{\text{random}}(x, z) \) to be the above computation except that \( L^{(i)} \) is chosen uniformly at random in step 1. Note that \( \mathcal{R}_{\text{random}} \) is a distribution on 1-local functions.

To prove the lemma, we define a set of hybrid distributions \( H_m \) on the wires of \( \text{random} \) for \( m = 1, \ldots, t - 1 \). Fix any plausible \( x, z \in G^t \). In \( H_m \), the wires in the tree computing \( z_i \) for \( i \leq m \) are chosen as in \( W_{x \to z} \), and for \( i > m \) the wires are chosen as in \( \mathcal{R}_{\text{random}}(x, z) \). Then, we have \( H_1 \equiv \mathcal{R}_{\text{random}}(x, z) \) and \( H_{t-1} \equiv W_{x \to z} \) (note that the first and last trees are distributed identically in \( W_{x \to z} \) and \( \mathcal{R}_{\text{random}}(x, z) \)). Thus if there is a function \( \ell \) such that

\[
\Delta(\ell(\mathcal{R}_{\text{random}}(x, z)), \ell(W_{x \to z})) > \epsilon \cdot (t-1)
\]

then there is an \( m \in [2, t - 1] \) such that

\[
\Delta(\ell(H_{m-1}), \ell(H_m)) > \epsilon. \tag{3.6}
\]

Now let \( g \in G \) be the fixed value (depending on \( x, z \)) such that \( \prod_j R^{(m-1)}_j = g \) with probability 1 in both \( H_m \) and \( H_{m-1} \). Thus in \( H_{m-1} \) (resp. \( H_m \)), \( L^{(m)} \) is distributed according to \( U_{G^t} \) (resp. \( D_{g^{-1}} \)). (Note that this \( g \) is arbitrary, i.e. not only \( \alpha \) or \( \text{id} \), and hence we are using the full generality of Definition 3.1.2.) Because \( H_{m-1} \) and \( H_m \) differ only in the tree computing \( z_m \), and because the distribution on these wires is independent of all other wires when \( x \) and \( z \) are fixed, by an averaging argument we can fix all wires outside this tree while preserving (3.6). Then given a vector \( v \in G^t \) distributed according to either \( U_{G^t} \) or \( D_{g^{-1}} \), a 1-local function
(of \(v\)) can generate \(H_{m-1}\) or \(H_m\) by setting \(L^{(m)} := v\) and performing steps 2-4 in the above computation. Fixing the randomness of this function by an averaging argument, we obtain the function \(f : G^t \rightarrow G^{[\text{RANDOM]}\text{]}\) in the statement of the lemma.

Using the above gadgets and Lemma 3.2.1, we now complete the proof of Theorem 3.1.4 (restated for convenience) essentially following [FRR+10, Lemma 13].

**Theorem 3.1.4.** For every polynomial-time computable function \(t = t(n)\) and every sequence of groups \(G = G(t)\) parameterized by \(t\), there is a compiler \(\text{Comp}\) for which the following holds.

1. For every \(C : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n\) and \(k \in \{0,1\}^n\), \(\text{Comp}(C,k)\) runs in time \(\text{poly}(|C|, t, \log |G|)\) and outputs a circuit \(\hat{C}\) of size \(O(|C| \cdot t^2 \cdot \log |G|)\).

2. For every set of functions \(L\) and every \(\epsilon > 0\), if the 4-local extension of \(L\) is \(\epsilon\)-fooled by \(G^t\) then \(\text{Comp}\) is an \((L, \epsilon \cdot t \cdot |C|)\)-leakage-secure compiler.

**Proof.** Let \(C : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n\) and \(k \in \{0,1\}^n\) be the input to \(\text{Comp}\). As described in §3.2, \(\text{Comp}\) constructs a circuit \(\hat{C}\) by replacing each wire in \(C\) with a bundle of \(t\) wires, and replacing each gate in \(C\) with a set of gadgets. Specifically for each Nand gate in \(C\) with two input wires and \(m\) output wires, \(\hat{C}\) contains a NAND gadget followed by \(m\) RANDOM gadgets in parallel (each of which takes as input the NAND gadget’s output).

In order for \(\hat{C}(x)\) to map \(\{0,1\}^n \rightarrow \{0,1\}^n\), it must encode \(x \in \{0,1\}^n\) to \(x' \in (G^t)^n\) as a first step, and decode \(z' \in (G^t)^n\) to \(z \in \{0,1\}^n\) as a final step. This is done in the following straightforward way. The input encoder sets each \(x'_i\) to be either \((\text{id}, \ldots, \text{id})\) or \((\text{id}, \ldots, \text{id}, \alpha)\) depending if \(x_i = 0\) or \(1\). The output decoder computes each product \(\prod_{j=1}^t (z'_i)_j\), and sets \(z_i = 0\) or \(1\) depending if this product is \(\text{id}\) or \(\alpha\). The decoder may use any correct multiplication tree, i.e. the specific tree used is not relevant to the proof of security.

The size bound on \(\text{Comp}\) is immediate. To prove that \(\hat{C}\) is a correct circuit (i.e. that \(\hat{C}(x) = C(x,k)\) for every \(x\)), one can apply an inductive argument to show that each bundle at the output of a RANDOM gadget correctly encodes the value of the corresponding wire in \(C\), and thus the output decoder indeed produces \(C(x,k)\).

In the hybrid arguments below, we will crucially use the fact that each bundle of the secret state and each bundle at the output of a RANDOM gadget is uniform (over the random coins of \(\text{Comp}\)) subject to correctly encoding the corresponding wire of \(C\).

On input \((C,x,\hat{C}(x))\), the simulator \(S\) computes a distribution on the wires of \(\hat{C}\) as follows.
First, $S$ computes the wires for the encoder and decoder honestly. For the encoder this is straightforward. For the decoder, $S$ chooses $n$ vectors $z'_i \in G^t$ which are uniform conditioned on the correct product (determined by $\hat{C}(x)_i$), and then computes the wires for the multiplication trees honestly. These wires are distributed identically to the real distribution on $\hat{C}(x)$’s wires and thus will not affect the hybrid arguments that follow, which is why these multiplication trees and the complexity of $S$ for this step are not of interest.

Next, $S$ chooses uniformly at random the values for each wire encoding the secret input $k$, as well as each connecting wire at the output of a random gate (except those which touch the output decoder and have already been chosen).

Next, for each NAND gadget $S$ computes values for its internal wires and for its output wires by simply evaluating the gadget. (Here we use the fact that the output of one NAND gadget is never the input of another, so all NAND input bundles have already been set.)

Finally, $S$ computes internal wire values for each random gadget using $R_{\text{random}}$.

Now let $\hat{C} \leftarrow \text{Comp}(C, k)$, and recall that $\hat{W}_x$ denotes the real distribution on the wires of $\hat{C}(x)$. We define an intermediate distribution $W'_x$ as follows: first draw a sample from $\hat{W}_x$, and then recomputing the internal wires of each random gadget from its input/output bundles using $R_{\text{random}}$.

We now show that $W'_x$ is indistinguishable (by $L$) from both $\hat{W}_x$ and $S(C, x, \hat{C}(x))$.

**Claim 3.** If the 1-local extension of $L$ is $\epsilon$-fooled by $G^t$, then $\forall \ell \in L$:

$$\Delta(\ell(W'_x), \ell(\hat{W}_x)) \leq \epsilon \cdot |C| \cdot (t-1).$$

**Proof.** Assume that there an $\ell \in L$ such that $\Delta(\ell(W'_x), \ell(\hat{W}_x)) > \epsilon \cdot |C| \cdot (t-1)$. Define some fixed ordering on the $\leq |C|$ random gadgets of $\hat{C}$. Then by a hybrid argument, there is an $m \leq |C|$ and two distributions $H$ and $H'$, defined as follows, for which $\Delta(\ell(H), \ell(H')) > \epsilon \cdot (t-1)$. $H$ is defined by first drawing a sample from $\hat{W}_x$, and then recomputing the internal wires of random gadgets $1, \ldots, m$ from their input/output bundles; $H'$ is the same except only random gadgets $1, \ldots, m-1$ are recomputed.

Now by an averaging argument, we can fix all wires in both $H$ and $H'$ except those internal to the $m$th random gadget, obtaining a function $\ell'$ (with domain $G^{|\text{random}|}$) in the 1-local extension of $L$. Then $\ell'$ distinguishes the real wires of the $m$th random gadget from those computed by $R_{\text{random}}$ with advantage $> \epsilon \cdot (t-1)$. In combination with Lemma 3.2.1, this contradicts the claim’s hypothesis.

\qed
Claim 4. If the 4-local extension of $L$ is $\epsilon$-fooled by $G^t$, then $\forall \ell \in L$:

$$\Delta(\ell(S(C, x, \hat{C}(x))), \ell(W'_x)) \leq \epsilon \cdot |C|.$$ 

Proof. Assume that there an $\ell \in L$ such that $\Delta(\ell(S(C, x, \hat{C}(x))), \ell(W'_x)) > \epsilon \cdot |C|$. Define some fixed ordering on the $\leq |C|$ bundles of $\hat{C}$ that either encode a bit of the secret input $k$ or are at the output of a random gadget but do not touch the output decoder. Then by a hybrid argument, there is an $m \leq |C|$ and two distributions $H$ and $H'$, defined as follows, for which $\Delta(\ell(H), \ell(H')) > \epsilon$. In $H$, bundles $1, \ldots, m$ are uniformly random and bundles $m + 1, \ldots, |C|$ are random subject to correctly encoding the value of the corresponding wire in $C$; in $H'$ only bundles $1, \ldots, m - 1$ are uniformly random. In both, each NAND’s internal wires are computed using the gadget itself, and each RANDOM’s internal wires are computed using $R_{\text{random}}$.

Let $g \in \{\text{id}, \alpha\}$ be the value encoded by the $m$th bundle in $W'_x$ (determined by $C, k$ and $x$). Note that the $m$th bundle is necessarily the input of a NAND gadget, and is either the output of a random gadget or a bundle encoding a bit of $k$. By an averaging argument, we can fix all wires in $H$ and $H'$ while preserving $\Delta(\ell(H), \ell(H')) > \epsilon$, except for the following: the $m$th bundle, the internal and output wires of the NAND gadget that it touches, the internal wires of the RANDOM gadgets that are adjacent to the output of this NAND gadget, and the internal wires of the RANDOM gadget that outputs the $m$th bundle (if it exists).

Finally, a 4-local function can compute one of the two distributions from an input $v \in G^t$ distributed according to either $U_{G^t}$ or $D_g$: it plugs $v$ into the $m$th bundle and computes the (4-local) NAND gadget and the (1-local) $R_{\text{random}}$. 

Finally, because $\Delta$ is a metric, these two claims give part 2 of the theorem.

3.3 Compressing group products

In this section we show that $(A_u)^t$ fools in the sense of Definition 3.1.2 a number of computational models, proving Theorem 3.1.5. As mentioned previously, the results of this section can be viewed as giving compression bounds against a variety of classes for the task of computing $A_u$-products. We start by recalling a number of facts related to groups and computation. Then in each of the subsections we analyze each computational model in turn.

First, it will be convenient later to prove that $\ell(D_\alpha)$ and $\ell(D_\text{id})$ are close for every $\alpha \in G$, $\ell \in L$, and we observe that this is sufficient.
Lemma 3.3.1. If $\Delta(\ell(D_\alpha), \ell(D_\beta)) \leq \epsilon$ for every $\alpha \in G$ and every $\ell$ in the 1-local extension of $L$, then $L$ is $\epsilon$-fooled by $G^\ell$.

Proof. If $L$ is not $\epsilon$-fooled by $G^\ell$ then $\exists \alpha \in G$, $\ell \in L$ such that $\Delta(\ell(D_\alpha), \ell(U_{G^\ell})) > \epsilon$. Then by an averaging argument, $\exists \beta \in G$ such that $\Delta(\ell(D_\alpha), \ell(D_{\beta})) > \epsilon$. Defining $\ell'(x_1, \ldots, x_t) := \ell(x_1, \ldots, x_t \cdot \beta)$ in the 1-local extension of $L$, we have $\Delta(\ell'(D_{\alpha\beta-1}), \ell'(D_\beta)) > \epsilon$. \hfill \Box

We will also make use of the random self-reducibility of the distributions $D_\alpha$.

Lemma 3.3.2 ([Kil88]). There exists a distribution on 1-local functions $R : G^\ell \to G^\ell$ such that for any $\alpha \in G$ and any $x$ in the support of $D_\alpha$, $R(x) \equiv D_\alpha$.

Proof. $R$ chooses $r_1, \ldots, r_{t-1} \in G$ uniformly and outputs $(x_1r_1, r_1^{-1}x_2r_2, \ldots, r_{t-1}^{-1}x_t)$. \hfill \Box

Recall the following standard terminology: $\alpha$ is an involution if $\alpha = \alpha^{-1}$, and $\alpha$ is the commutator of $\beta$ and $\gamma$ if $\alpha = \beta\gamma\beta^{-1}\gamma^{-1}$. We say that $M : \{0,1\}^n \to (A_5)^\ast$ $\alpha$-computes a function $f : \{0,1\}^n \to \{0,1\}$ if for every $x$, $\prod_i M(x)_i = \alpha^{f(x)}$ (where $\alpha^0 = \text{id}$ and $\alpha^1 = \alpha$).

The next theorem follows from [Bar89, Theorem 5] because every element of $A_5$ is a commutator.

Theorem 3.3.3 ([Bar89]). For every $\alpha \in A_5$ and every fan-in-2 circuit $C : \{0,1\}^n \to \{0,1\}$ of depth $d$, there is a 1-local function $M : \{0,1\}^n \to (A_5)^{O(4^d)}$ that $\alpha$-computes $C$.

Moreover, let $f_1, \ldots, f_m : \{0,1\}^n \to \{0,1\}$ be functions such that for each $i \leq m$ there is a 1-local function $M'_i : \{0,1\}^n \to (A_5)^n$ that $\alpha$-computes $f_i$. Then, for every fan-in-2 circuit $C : \{0,1\}^m \to \{0,1\}$ of depth $d$ there is a 1-local function $M : \{0,1\}^mn \to (A_5)^{O(n \cdot 4^d)}$ that $\alpha$-computes $C(f_1(\cdot), \ldots, f_m(\cdot))$.

The following two lemmas allow certain functions to be more efficiently $\alpha$-computed. These can be compared with the works by Cai and Lipton [CL94] and Cleve [Cle91] which give increasingly efficient versions of Barrington’s construction (here efficiency is measured in the length of $M$’s output). Our construction is simpler than the ones given in these works, but also less general.

Lemma 3.3.4. For every involution $\alpha \in A_5$, the following holds.

1. There is a 1-local function $M : \{0,1\}^n \to (A_5)^n$ that $\alpha$-computes $\bigoplus_{i=1}^nx_i$.

2. If $M$ and $M'$ $\alpha$-compute some Boolean functions $f$ and $f'$, then their concatenation $(M, M')$ $\alpha$-computes the function $f \oplus f'$. 
Proof. For the first item: given input $x \in \{0, 1\}^n$, output $y \in (A_5)^n$ such that $y_i = \alpha^{x_i}$. The correctness of this construction, as well as the second item, follows from the isomorphism between the group $\{0, 1\}$ (under $\oplus$) and the subgroup $\{\text{id}, \alpha\} \subset A_5$.

For the next lemma, note that because every $k$-cycle $(a_1 a_2 \cdots a_k)$ can be written as a product of $k-1$ transpositions $(a_1 a_2)(a_1 a_3) \cdots (a_1 a_k)$, every element of $A_5$ is either a 3-cycle, a 5-cycle, the product of two disjoint transpositions, or the identity.

**Lemma 3.3.5.** Every non-involution in $A_5$ is the commutator of two involutions.

**Proof.** Let $a, b, c, d, e$ denote arbitrary, distinct elements of $\{1, \ldots, 5\}$. Every non-involution in $A_5$ is either a 3-cycle $(a b c)$ which is the commutator of involutions $(a b)(d e)$ and $(b c)(d e)$, or a 5-cycle $(a b c d e)$ which is the commutator of involutions $(b e)(c d)$ and $(a d)(b c)$.

### 3.3.1 Multi-party protocols

In this section we consider functions computable by a multi-party communication protocol in the “number on forehead” model [CFL83], defined as follows. A protocol $P$ with $n$-bit inputs consists of $k = k(n)$ parties, each with unlimited computational power. The input $x \in \{0, 1\}^n$ is partitioned into $k$ blocks, and party $i$ sees all input bits except those in the $i$th block. The parties communicate in the broadcast model, so every bit sent is seen by all parties. The $(m = m(n))$-bit output of $P$ is defined to be the final $m$ bits that are broadcast, and the cost of $P$ is the total number of bits broadcast by all parties. When $P$’s input comes from a group $G$, we assume some canonical representation of $G$’s elements as $\log |G|$-bit strings.

We prove the following compression bound for such protocols.

**Theorem 3.3.6.** There is a partition of the inputs in $(A_5)^t$ into $k$ pieces such that any $k$-party protocol communicating $c$ bits and outputting $\leq c$ bits is $\epsilon$-fooled by $(A_5)^t$ for $\epsilon = 2^{-\Omega(t/(k^2 4^k))}$.

We prove this theorem by combining an efficient translation from bits to group products with the following lower bound. Define the generalized inner-product function $GIP_{n,k} : \{0, 1\}^{nk} \to \{0, 1\}$ as

$$GIP_{n,k}(x) := \bigoplus_{i=1}^{n} \bigwedge_{j=1}^{k} x_{i,j}.$$  

Then we have the following lemma, originally due to Babai, Nisan, and Szegedy [BNS92] and with increasingly streamlined proofs in [CT93, Raz00, VW08].
Lemma 3.3.7 ([BNS92]). There is a partition of the inputs to $GIP_{n,k}$ into $k$ blocks such that for every protocol $P : \{0,1\}^{nk} \rightarrow \{0,1\}$ with $k$ parties that communicates at most $c$ bits, $\Pr_x[P(x) = GIP_{n,k}(x)] \leq \frac{1}{2} + 2^{c-\Omega(n/k)}$.

We give the following translation to bits from group products.

Lemma 3.3.8. For every $\alpha \in A_5$, there is a 1-local function $M : \{0,1\}^{nk} \rightarrow (A_5)^{O(n^2k)}$ that $\alpha$-computes $GIP_{n,k}$.

Proof. Assume $\alpha$ is an involution. Let $M' : \{0,1\}^k \rightarrow (A_5)^{O(k^2)}$ be the function guaranteed by Theorem 3.3.3 that $\alpha$-computes the $k$-wise AND of its input. Then letting $x^{(i)} := (x_{i,1}, \ldots, x_{i,k})$ for each $i \leq n$, the function

$$M(x) := (M'(x^{(1)}), \ldots, M'(x^{(n)}))$$

$\alpha$-computes $GIP_{n,k}$ by the second item of Lemma 3.3.4.

If $\alpha$ is not an involution, let $\beta, \gamma \in A_5$ be the involutions guaranteed by Lemma 3.3.5 such that $\alpha = \beta \gamma \beta \gamma$ (note that $\beta = \beta^{-1}$ and $\gamma = \gamma^{-1}$). Then let $M'$ instead $\beta$-compute the AND of its input, and compute $M$ as

$$M(x) := (M'(x^{(1)}), \ldots, M'(x^{(n)}), \gamma, M'(x^{(1)}), \ldots, M'(x^{(n)}), \gamma)$$

We now give the proof of Theorem 3.3.6.

Proof of Theorem 3.3.6. For an appropriate $n = \Omega(t/k^2)$, let $M : \{0,1\}^{n \cdot k} \rightarrow (A_5)^t$ be the 1-local function guaranteed by Lemma 3.3.8 that $\alpha$-computes $GIP_{n,k}$. Consider the partition on the $t$ elements of the input from $(A_5)^t$ that is induced by $M$ from the partition on $GIP_{n,k}$ guaranteed by Lemma 3.3.7.

Assume for contradiction that some protocol on this partition is not $\epsilon$-fooled. Without loss of generality the protocol outputs 1 bit. (The last player can simulate whatever set maximizes the statistical distance of the multi-bit protocol output distributions.) By Lemma 3.3.1, there is an $\alpha \in A_5$ and a protocol $P : (A_5)^t \rightarrow \{0,1\}$ with $k$ parties communicating $\leq c$ bits such that

$$\Delta(P(D_\alpha), P(D_\beta)) \geq 2^{c-\beta t/(k^2k^k)}$$

for a suitable constant $\beta$.

By combining the $P$ with $M$ we now give a distribution on protocols $P' : \{0,1\}^{n \cdot k} \rightarrow \{0,1\}$ for $GIP_{n,k}$ with the same number of parties, the same communication, and the same advantage up to the constant $\beta$. This contradicts Lemma 3.3.7.
On input $x$, each party in $P'$ first computes the portion of $y := M(x) \in (A_5)^t$ that depends on the input bits it can see; this is done with no communication as $M$ is 1-local. Next each party computes the portion of $z := (y_1 \cdot r_1 \cdot y_2 \cdot r_2 \cdot \ldots \cdot y_i r_i \cdot \ldots \cdot y_t r_t)$ that depends on the input bits it can see, again with no communication. ($r$ is a public random string.) Finally the parties compute and output $P(z)$ using the protocol $P$.

Note for every $x$ such that $GIP_{n,k}(x) = 1$ (resp. $GIP_{n,k}(x) = 0$, $z$ is distributed according to $D_\alpha$ (resp. $D_\alpha$) over the choice of $r$. The proof is now completed using the fact that $\Pr_x[GIP_{n,k}(x) = 0] > \Pr_x[GIP_{n,k}(x) = 1] \geq 1/2 - 2^{-\Omega(n/2k)}$ [VW08, Claim 2.11].

3.3.2 $AC^0$

In this section, we observe that by combining the following compression bound against $AC^0$ due to Dubrov and Ishai [DI06] with our Lemma 3.3.4 above, we can obtain a quantitatively identical compression bound for $A_5$-products against $AC^0$.

Let $\oplus^{-1}(b)$ denote the uniform distribution over $n$-bit strings with parity = $b$.

**Theorem 3.3.9 ([DI06]).** For every $0 < \delta < 1$ and every $d \in \mathbb{N}$, there is a constant $\epsilon > 0$ such that the following holds. For every unbounded-fan-in circuit $C : \{0,1\}^n \to \{0,1\}^t$ of depth \leq $d$ and size $\leq 2^{n(1-\delta)/d}$:

$$\Delta(C(\oplus^{-1}(0)), C(\oplus^{-1}(1))) < 2^{-n(1-\delta)/d}.$$  

**Theorem 3.3.10.** For every $0 < \delta < 1$ and every $d \in \mathbb{N}$, there is a constant $\epsilon > 0$ such that the following holds. Let $\mathcal{L}$ be the class of unbounded-fan-in circuits $C : (A_5)^t \to \{0,1\}^d$ of depth $\leq d$ and size $\leq 2^{t(1-\delta)/d}$. Then, $\mathcal{L}$ is $2^{-\epsilon t(1-\delta)/d}$-fooled by $(A_5)^t$.

**Proof.** Assume for contradiction that $\mathcal{L}$ is not $2^{-\epsilon t(1-\delta)/d}$-fooled by $(A_5)^t$. Then by Lemma 3.3.1 there is a circuit $C \in \mathcal{L}$ and an $\alpha \in A_5$ such that

$$\Delta(C(D_\alpha), C(D_\alpha)) \geq 2^{-\epsilon t(1-\delta)/d}.$$  

For $n = \Omega(t)$, let $M : \{0,1\}^n \to (A_5)^t$ be the 1-local function guaranteed by Lemmas 3.3.4 and 3.3.5 such that

$$\prod_{i=1}^{t} M(x)_i = \begin{cases} id & \text{if } \bigoplus_{j=1}^{n} x_j = 0 \\ \alpha & \text{if } \bigoplus_{j=1}^{n} x_j = 1. \end{cases}$$  

Let $R$ be the function from Lemma 3.3.2. Then we have

$$\Delta(C(R(M(\oplus^{-1}(0)))), C(R(M(\oplus^{-1}(1))))) \geq 2^{-\epsilon t(1-\delta)/d} = 2^{-\epsilon \cdot n/(d+1)}.$$
for appropriately chosen $\epsilon' = \epsilon'(\epsilon)$ and $\delta' = \delta'(\delta)$. Noting that the depth of $C(R(M(\cdot)))$ is $d + 1$ and using an averaging argument to fix the randomness of $R$, this contradicts Theorem 3.3.9. □

3.3.3 AC^0 with symmetric gates

Here we show that our hardness assumption holds when $\mathcal{L}$ is the class of unbounded-fan-in constant-depth circuits that contain $t^{O(\log t)}$ And/Or/Not gates and $O(\log^2 t)$ gates that each compute an arbitrary symmetric function. Specifically, we prove the following.

**Theorem 3.3.11.** For every $d$, there is an $\epsilon > 0$ such that the following holds for every $t$.

Let $\mathcal{L}$ be the set of functions $\ell : (A_5)^t \rightarrow \{0,1\}^{t^{0.1}}$ where each output bit of $\ell$ is computable by an unbounded-fan-in circuit of depth $\leq d$ that contains $\leq t^\log t$ And/Or/Not gates and $\leq \epsilon \log^2 t$ arbitrary symmetric gates.

Then, $\mathcal{L}$ is $t^{-\epsilon \log t}$-fooled by $(A_5)^t$.

Note that in fact these circuits have up to $O(t^{0.1} \cdot \log^2 t)$ arbitrary symmetric gates, though each output bit only depends on $O(\log^2 t)$ of them.

The proof of Theorem 3.3.11 extends a lower bound due to Viola [Vio07] and combines it with an efficient translation from bits to group products. Viola’s lower bound shows that the following function $PAP_{n,m} : \{0,1\}^{n^2 m} \rightarrow \{0,1\}$ (for “parity-and-parity”) is hard on average for this circuit class to compute.

$$PAP_{n,m}(x) := \bigoplus_{i=1}^{n} \bigwedge_{j=1}^{m} \bigoplus_{k=1}^{n} x_{i,j,k}$$

We use the following translation from $PAP$-inputs to group products.

**Theorem 3.3.12.** For every $\alpha \in A_5$, there is a 1-local function $M : \{0,1\}^{n^2 m} \rightarrow (A_5)^{O(n^2 m^2)}$ that $\alpha$-computes $PAP_{n,m}$.

The proof of this theorem is analogous to that of Lemma 3.3.8, using the “moreover” part of Theorem 3.3.3. We omit the details.

For the remainder of this section, we let $PAP$ denote $PAP_{n,0.3 \log n}$. By combining Theorem 3.3.12 with the random self-reducibility of the distributions $D_\alpha$, we prove the following.

**Lemma 3.3.13.** Let $\mathcal{L}$ be as in Theorem 3.3.11, and assume that $\mathcal{L}$ is not $t^{-\epsilon \log t}$-fooled by $(A_5)^t$. Then there is an $n = \Omega(\sqrt{t}/\log t)$, an $\epsilon' = \epsilon'(\epsilon) > 0$, and a function $\ell' : \{0,1\}^{n^2 0.3 \log n} \rightarrow \{0,1\}^{t^{0.1}}$ such that each output bit of $\ell'$ is computable by an unbounded-fan-in circuit of depth
\[ \Delta(\ell'(\text{PAP}^{-1}(1), \ell'(\text{PAP}^{-1}(0))) \geq n^{-\epsilon' \log n}. \]

**Proof.** If \( \mathcal{L} \) is not \( t^{-\epsilon \log t} \)-fooled by \((A_5)^t\), then there is an \( \alpha \in A_5 \) and an \( \ell \in \mathcal{L} \) such that \[ \Delta(\ell(D_\alpha), \ell(D_{\text{id}})) \geq t^{-\epsilon t}. \] For an appropriate \( n = \Omega(\sqrt{t} \log t) \), let \( M : \{0,1\}^{n^2 \cdot 0.3 \log n} \rightarrow (A_5)^t \) be the 1-local function guaranteed by Theorem 3.3.12 that \( \alpha \)-computes PAP. Let \( R : (A_5)^t \rightarrow (A_5)^t \) be the randomized 1-local function from Lemma 3.3.2. Then \( \ell' := \ell(R(M(\cdot))) \) satisfies the lemma, and by an averaging argument we can fix the randomness of \( R \) so that \( \ell' \) is deterministic.

We now prove Theorem 3.3.11. The lower bound in \([\text{Vio07}]\) shows that, when restricted to one output bit and with domain \( \mathbb{B} = \{0,1\}^{n^2 \cdot 0.3 \log n} \), circuits in \( \mathcal{L} \) have correlation \( n^{-\Omega(\log n)} \) with PAP. This is done by showing that with probability \( 1 - n^{-\Omega(\log n)} \) over a random restriction \( \rho \) to the input bits of a circuit \( C \in \mathcal{L} \), we have both that \( \text{PAP}|\rho = \text{GIP} \) and that \( C|\rho \) is computable by a \((0.3 \log n)\)-party protocol communicating \( \log^5 n \) bits, which triggers the lower bound of Lemma 3.3.7.

In addition to the translation to group products above, we extend this argument to \( t^{0.1} \) output bits by using a union bound to show that \( \rho \) satisfies these properties simultaneously for all output bits, again with probability \( 1 - n^{-\Omega(\log n)} \). The protocol now exchanges \( t^{0.1} \cdot \log^5 n < n^{0.21} \) bits which is still sufficiently small to use Lemma 3.3.7.

**Proof of Theorem 3.3.11.** Assume that \( \mathcal{L} \) is not \( t^{-\epsilon \log t} \)-fooled by \((A_5)^t\), and let \( n, \epsilon', \) and \( \ell' = (\ell'_1, \ldots, \ell'_{t^{0.1}}) \) be given by Lemma 3.3.13. Let \( R \) be the following distribution over restrictions \( \rho \) on \( n^2 \cdot 0.3 \log n \) bits that leave \( n \cdot 0.3 \log n \) bits unrestricted:

- Choose \( \rho' \) uniformly over all restrictions that leave \((n^2 \cdot 0.3 \log n)^{0.9} \) bits unset.

- If \( \text{PAP}|\rho \) has \( \geq 1 \) input unrestricted per bottom \( \oplus \) gate, then choose \( \rho'' \) uniformly over restrictions to the remaining bits that leave exactly 1 input unrestricted per bottom \( \oplus \) gate.

- Else, choose \( \rho'' \) uniformly over all restrictions to the remaining bits that leave exactly \( n \cdot 0.3 \log n \) bits unrestricted.

- Output \( \rho = \rho' \circ \rho'' \).
Say that \( \rho \) is good if \( \text{PAP}_\rho \) has exactly 1 input unrestricted per bottom \( \oplus \) gate and for every \( i = 1, \ldots, t^{0.1} \), \( \ell'_i[\rho] \) is computable by a \((0.3 \log n)\)-party protocol (under any partitioning of the input) exchanging \( \log^5 n \) bits of communication. Combining [Vio07, Claim 11 & Lemma 12] with a union bound over all \( \ell'_i \), we obtain

\[
\Pr_{\rho \leftarrow R} [\rho \text{ is good}] \geq 1 - n^{-\Omega(\log n)}.
\]

Because \( \Delta(\ell'(\text{PAP}^{-1}(1)), \ell'(\text{PAP}^{-1}(0))) \geq n^{-\epsilon' \log n} \), there is a set \( S \subseteq \{0,1\}^{t^{0.1}} \) such that

\[
\Pr_x [\ell'(x) \in S \mid \text{PAP}(x) = 1] - \Pr_x [\ell'(x) \in S \mid \text{PAP}(x) = 0] \geq n^{-\epsilon' \log n}. \tag{3.7}
\]

For any \( \rho \) that is good, let \( P_\rho : \{0,1\}^{(0.3 \log n)} \to \{0,1\} \) be the following \((0.3 \log n)\)-party protocol exchanging \( t^{0.1} \cdot \log^5 n + 1 \leq n^{0.21} \) bits. On input \( y \), the parties first compute each \( \ell'_{i|\rho}(y) \) by communicating \( t^{0.1} \cdot \log^5 n \) bits, and then output 1 iff \( \ell'_{|\rho}(y) \in S \) using one additional bit of communication.

For every \( \rho \) that is good, \( \text{PAP}_\rho \) is equal (up to complementing some inputs) to the generalized inner-product function \( \text{GIP} \) from §3.3.1. Thus, by Lemma 3.3.7 we have

\[
\Pr_y [P_\rho(y) = \text{PAP}_\rho(y)] < 1/2 + 2^{-n^{\Omega(1)}} \tag{3.8}
\]

for every good \( \rho \).

Now notice that choosing a random \( x \in \{0,1\}^{n^2 - 0.3 \log n} \) can be thought of as first choosing \( \rho \) from \( R \) and then choosing \( y \) uniformly over \( \{0,1\}^{n - 0.3 \log n} \). Then letting \( E_b \) denote the event “\( \rho \) is good and \( \text{PAP}_\rho(y) = b \)”, we have

\[
\Pr_x [\ell'(x) \in S \mid \text{PAP}(x) = 1] - \Pr_x [\ell'(x) \in S \mid \text{PAP}(x) = 0]
\]

\[
= \Pr_{\rho,y} [\ell'_{\rho}(y) \in S \mid \text{PAP}_\rho(y) = 1] - \Pr_{\rho,y} [\ell'_{\rho}(y) \in S \mid \text{PAP}_\rho(y) = 0]
\]

\[
\leq \Pr_{\rho,y} [\ell'_{\rho}(y) \in S \mid E_1] - \Pr_{\rho,y} [\ell'_{\rho}(y) \in S \mid E_0] + \Pr_{\rho} [\rho \text{ is not good}]
\]

\[
= \Pr_{\rho,y} [P_\rho(y) = 1 \mid E_1] - \Pr_{\rho,y} [P_\rho(y) = 1 \mid E_0] + \Pr_{\rho} [\rho \text{ is not good}]
\]

\[
= \Pr_{\rho,y} [P_\rho(y) = 1 \mid E_1] + \Pr_{\rho,y} [P_\rho(y) = 0 \mid E_0] - 1 + \Pr_{\rho} [\rho \text{ is not good}]
\]

\[
< (1/2 + 2^{-n^{\Omega(1)}})/(1/2 - 2^{-n^{\Omega(1)}}) - 1 + \Pr_{\rho} [\rho \text{ is not good}]
\]

\[
= 2^{-n^{\Omega(1)}} + \Pr_{\rho} [\rho \text{ is not good}]
\]

\[
\leq n^{-\Omega(\log n)}
\]

which contradicts (3.7) for sufficiently small \( \epsilon' \). Note that the second inequality follows from (3.8) because \( \text{PAP}_\rho = \text{GIP} \) is balanced up to an additive factor of \( 2^{-n^{\Omega(1)}} \).

\( \Box \)
3.3.4 TC$^0$

In this section we show that, because computing products over $A_5$ is complete for NC$^1$, if TC$^0 \neq$ NC$^1$ then the set of TC$^0$ circuits with $O(\log t)$ bits of output is $t^{-\omega(1)}$-fooled by $(A_5)^t$.

Recall that NC$^1$ is the class of poly-size fan-in-2 And/Or/Not circuits with depth $O(\log n)$, and TC$^0$ is the class of poly-size unbounded-fan-in constant-depth circuits where each gate computes, for some $c$, the $c$-threshold function which is 1 iff $c$ inputs are 1.

The high-level idea behind the next theorem is the following. Assume there is an $A_5$ and a TC$^0$ circuit $C$ that can distinguish between $D_\alpha$ and $D_{id}$ with advantage $\geq t^{-k}$ for some $k$. Then for any NC$^1$ circuit $B$ we construct a TC$^0$ circuit that, on input $x$, chooses $m = t^{O(k)}$ samples from the distribution $D_{\alpha^B(x)}$ and outputs 1 iff the number of samples on which $C$ outputs 1 is sufficiently close to $m \cdot \Pr[C(D_\alpha) = 1]$. This last check can be computed with threshold gates, and we can sample from $D_{\alpha^B(x)}$ by using Theorem 3.3.3 to obtain a single element in its support and then relying on the random self-reducibility of this distribution.

**Theorem 3.3.14.** If TC$^0 \neq$ NC$^1$ then $\forall k$ and infinitely many $t$, the class $L$ of TC$^0$ circuits with size $\leq t^k$ and output length $k \log t$ is $t^{-k}$-fooled by $(A_5)^t$.

*Proof.* Assume there is an $\alpha \in A_5$ and a TC$^0$ circuit $C$ such that $\Delta(C(D_\alpha), C(D_{id})) \geq t^{-k}$. Let $S \subseteq \{0, 1\}^{k \log t}$ be the set that maximizes $\Pr[C(D_\alpha) \in S] - \Pr[C(D_{id}) \in S]$, and note that checking $x \in S$ can be done by a TC$^0$ circuit of size $t^{O(k)}$. Thus, there is a TC$^0$ circuit $C' : (A_5)^t \to \{0, 1\}$ of size $t^{O(k)}$ such that

$$\Pr[C'(D_\alpha) = 1] - \Pr[C'(D_{id}) = 1] \geq t^{-k}. \quad (3.9)$$

Define $\epsilon_\alpha := \Pr[C'(D_\alpha) = 1]$ and $\epsilon_{id} := \Pr[C'(D_{id}) = 1]$, and note that $\epsilon_\alpha \geq t^{-k}$.

Let $B : \{0, 1\}^n \to \{0, 1\}$ be any NC$^1$ circuit, and for an appropriate $t = n^{O(1)}$ let $M : \{0, 1\}^n \to (A_5)^t$ be the 1-local function (guaranteed by Theorem 3.3.3) that $\alpha$-computes $B$. Let $C'' : \{0, 1\}^n \to \{0, 1\}$ be the randomized TC$^0$ circuit that performs the following steps on input $x \in \{0, 1\}^n$.

1. Compute $y = M(x) \in (A_5)^t$.

2. For $m := t^{5k}(n + 2)/\epsilon_\alpha = n^{O(k)}$, sample $z_1, \ldots, z_m \in (A_5)^t$ independently from $D_{\alpha^B(x)}$ by computing $R(y)$ where $R$ is the 1-local function from Lemma 3.3.2.
3. Use two layers of threshold gates to output 1 iff

\[ (1 - 1/(2t^k)) \cdot m\epsilon_\alpha \leq \sum_{i=1}^m C'(z_i) \leq (1 + 1/(2t^k)) \cdot m\epsilon_\alpha. \]

We now prove the following claim.

**Claim.** \( \forall x \in \{0,1\}^n : \Pr[C''(x) = B(x)] \geq 1 - 2^{-n-1} \) over the random coins of \( C'' \).

This implies the theorem, as follows. By a union bound there is a way to fix the random coins of \( C'' \) such that \( C''(x) = B(x) \) for every \( x \). Then because \( B \in \text{NC}^1 \) was arbitrary and \( C'' \in \text{TC}^0 \), we have \( \text{TC}^0 = \text{NC}^1 \).

**Proof of Claim.** Denote \( X := \sum_{i=1}^m C'(z_i), \) and \( \mu := \mathbb{E}[X]. \)

Fix \( x \), and first assume \( B(x) = 1 \) which means \( \mu = m\epsilon_\alpha = t^{3k}(n + 2) \). Then

\[ \Pr[C''(x) = B(x)] = \Pr \left[ |X - \mu| \leq \mu/(2t^k) \right] \geq 1 - 2e^{-\mu t^{-3k}} \geq 1 - 2^{-n-1} \]

by a Chernoff bound.

Now assume \( B(x) = 0 \). Then \( \mu = m\epsilon_{id} \), and since \( \epsilon_\alpha/\epsilon_{id} \geq 1 + 1/t^k \) by (3.9), we have

\[ (1 - 1/(2t^k)) \cdot m\epsilon_\alpha \geq \mu(1 + 1/(3t^k)) \]. Then using another Chernoff bound, we have

\[ \Pr[C''(x) = B(x)] \geq 1 - \Pr[X \geq \mu(1 + 1/(3t^k))] \geq 1 - e^{-\mu t^{-3k}} \geq 1 - 2^{-n-1}. \]

This completes the proof of the theorem.

### 3.3.5 \text{NC}^1

In this section we show that under the assumption \( \text{NC}^1 \neq \text{L} \), the set of \( \text{NC}^1 \) circuits with \( O(\log t) \) bits of output is \( t^{-\omega(1)} \)-fooled by \( (A_t)^i \).

**Theorem 3.3.15.** If \( \text{NC}^1 \neq \text{L} \) then for all \( k \) and infinitely many \( t \), the class of \( \text{NC}^1 \) circuits with depth \( k \log t \) and output length \( k \log t \) is \( t^{-k} \)-fooled by \( (A_t)^i \).

We would like to prove this analogously to Theorem 3.3.14 which was just proved. However, to do so we need an analog of Barrington’s theorem (Thm. 3.3.3) showing that the following notion of computing products over \( A_t \) is complete for \( \text{L} \).

**Definition 3.3.16.** Let \( G \) be a group. For \( \alpha \in G \) and \( t \in \mathbb{N} \), the \( \alpha \)-product problem over \( G^t \) is to decide, given \((x_1, \ldots, x_t) \in G^t \) such that \( \prod_i x_i \in \{\alpha, \text{id}\} \), which product it has.
Theorem 3.3.17. Assume that there is a circuit family $C$ of depth $O(\log t)$ such that for sufficiently large $t$, there exists $\alpha \in A_t$ such that $C$ decides the $\alpha$-product problem over $(A_t)^t$. Then $\text{NC}^1 = \text{L}$.

Theorem 3.3.15 is proved from Theorem 3.3.17 in exactly the same way that Theorem 3.3.14 is proved from Theorem 3.3.3. We omit the details, and now prove Theorem 3.3.17.

The **Cook-McKenzie construction.** Our starting point is the work by Cook and McKenzie [CM87] who show that a number of problems related to $S_t$, the group of all permutations on $t$ points, are complete for L. A key tool in their work is the following construction of a permutation that encodes acceptance of a given branching program on a given input. (We equate $\text{L}$ with the set of polynomial-size branching programs.)

**Theorem 3.3.18 ([CM87]).** There exists $t = O(s)$ and a circuit $C$ of depth $O(\log s)$ such that, on input $(B, x)$ where $B$ is a branching program of size $s \geq |x|$, $C$ outputs a permutation $\sigma \in S_t$ such that $B$ accepts $x$ iff 1 and $t$ are in the same cycle in $\sigma$’s disjoint cycle representation.

Recall that any permutation can be uniquely written as a product of disjoint cycles. Recall that we represent permutations $\sigma \in S_t$ using $(t \log t)$-bit pointwise representation as $(\sigma(1), \ldots, \sigma(t))$. Inversion and pairwise multiplication in $S_t$ can then be implemented in $\text{NC}^1$, because each essentially amounts to indexing an element from an array of length $t$.

Theorem 3.3.18 is proved by constructing a permutation $\sigma$ that performs one step of a depth-first search in $B$ when projecting to the edges consistent with the input $x$. The nodes are labeled by $[t]$, and the labels of the start and accept nodes (1 and $t$ wlog) are in the same cycle of $\sigma$ iff the accept node is reachable from the start node.

As observed to us by Eric Allender and V. Arvind (personal communication), Theorem 3.3.18 can be used to show that the following very natural problem is $\text{L}$-complete: given $(x_1, \ldots, x_t) \in (S_t)^t$, decide if $\prod_i x_i = \text{id}$. A simple embedding trick shows that this problem remains $\text{L}$-complete even over the group $A_t$ (see Theorem 3.3.22 below).

However it is not immediately clear how to use Theorem 3.3.18 to show that the $\alpha$-product problem is $\text{L}$-complete, for every or even for a single $\alpha \in A_t$. The issue is that the permutation $\sigma$ constructed in Theorem 3.3.18 depends on the structure of the branching program $B$ when projecting to the edges whose labels match the input $x$. So while the reduction to the “product $\neq \text{id}$” problem always produces a vector with product $= \text{id}$ when $B(x) = 0$, when $B(x) = 1$ the product can change depending on $x$, rather than being fixed to some $\alpha$. 
The proof of Theorem 3.3.17 has two steps. We first show that if NC\(^1\) can decide the \(\alpha\)-product problem for any fixed \(\alpha \in A_t\), then it can do so for every \(\alpha \in A_t\). Next we show that the \(\alpha\)-product problem is L-complete for the specific choice of \(\alpha = (1 2)(3 4)\), building on Theorem 3.3.18 and the extension by Allender and Arvind. The combination of these implies that if NC\(^1\) can decide the \(\alpha\)-product problem for any \(\alpha\), then NC\(^1\) = L.

### 3.3.5.1 Mapping group products

In this section we prove the following theorem.

**Theorem 3.3.19.** For all \(m, t \in \mathbb{N}\) and any \(\text{id} \neq \alpha, \beta \in A_t\), there is a 1-local function \(f : (A_t)^m \rightarrow (A_t)^{O(tm)}\) computable in depth \(O(\log t)\) that reduces the \(\alpha\)-product-problem to the \(\beta\)-product-problem, i.e. that satisfies \(\forall x = (x_1, \ldots, x_t) \in (A_t)^t:\)

\[
\prod_i x_i = \alpha \Rightarrow \prod_i f(x)_i = \beta \quad \text{and} \quad \prod_i x_i = \text{id} \Rightarrow \prod_i f(x)_i = \text{id}.
\]

To prove Theorem 3.3.15 we choose \(m = t\), but here we keep them separate to make the blowup in output length clear. The proof of Theorem 3.3.19 uses conjugation and commutation to construct the function \(f\). Recall from earlier that both of these operations are identity-preserving and can be computed locally. Namely we compute

\[
(x_1, \ldots, x_t) \mapsto (\gamma^{-1}x_1, \ldots, x_t\gamma)
\]

to map \(\alpha\)-products to \((\gamma^{-1}\alpha\gamma)\)-products and

\[
(x_1, \ldots, x_t) \mapsto (x_1, \ldots, x_t\gamma, x_t^{-1}, \ldots, x_1^{-1}\gamma^{-1})
\]

to map \(\alpha\)-products to \([\alpha, \gamma]\)-products. (We use the standard commutator notation \([\alpha, \gamma] := \alpha\gamma\alpha^{-1}\gamma^{-1}\).)

The following lemma shows that any \(\alpha \neq \text{id}\) can be converted to a product of two disjoint transpositions with \(\leq 2\) commutations.

**Lemma 3.3.20.** Let \(t \geq 4\). For every \(\text{id} \neq \alpha \in A_t\), there exist \(\gamma_1, \gamma_2 \in A_t\) such that \([\alpha, \gamma_1], \gamma_2\] is a double-transposition. Further, each \(\gamma_i\) is either a double-transposition or a 3-cycle.

**Proof.** We consider five cases based on \(\alpha\)’s cycle structure. In all cases except the last, in fact only one commutation is needed to obtain a double-transposition, but we can use two by noting that \([[(a b)(c d), (a b c)] = (a d)(b c)\].
1. If $\alpha$ contains a double-transposition $(a b)(c d)$, then $[\alpha, (a b c)] = (a d)(b c)$.

2. If $\alpha$ is a 3-cycle $(a b c)$, then $[\alpha, (a b)(c d)] = (a d)(b c)$.

3. If $\alpha$ contains two 3-cycles $(a b c)(d e f)$, then $[\alpha, (a d)(c f)] = (a d)(b e)$.

4. If $\alpha$ contains a 4-cycle $(a b c d)$, then $[\alpha, (a b)(c d)] = (a c)(b d)$.

5. If $\alpha$ contains a $(k \geq 5)$-cycle $(a_1 \cdots a_k)$, then $[\alpha, (a_2 a_3 a_4)] = (a_1 a_4 a_3)$ and so we can apply case 2.

Proof of Theorem 3.3.19. Assume $t \geq 10$ is even (similar techniques work for general $t$). First note that $\beta \in A_t$ can be written as the product of $t$ transpositions such that for each $1 \leq i \leq t/2$, the $(2i - 1)$-th and $2i$-th transpositions are disjoint. Without the disjointness requirement, $\beta$ can be written as a product of $t/2$ transpositions by writing it as a product of disjoint cycles and noting $(a_1 \cdots a_k) = (a_1 a_2) \cdots (a_1 a_k)$. Then if two adjacent transpositions overlap, we insert two copies of the transposition $(a b)$ between them where $a, b \leq t$ are present in neither of the adjacent transpositions. (This is where we use $t \geq 10$.) This satisfies the disjointness requirement, and preserves product $= \beta$ because the inserted transpositions cancel.

Now for each $1 \leq i \leq t/2$ let $\sigma_i$ be the product of the $(2i - 1)$-th and $2i$-th transpositions. Because $\alpha \neq \text{id}$, we can use $\leq 2$ commutations to convert it to a product of two disjoint transpositions via Lemma 3.3.20. Then because all products of two disjoint transpositions are conjugate in $A_t$, we can use 1 conjugation to convert it to $\sigma_i$. Because both commutation and conjugation are identity-preserving operations that can be computed locally as mentioned above, this gives a 1-local NC$^1$ function $f_i : (A_t)^t \to (A_t)^{4t}$ satisfying

$$\prod_j x_j = \alpha \Rightarrow \prod_j f_i(x)_j = \sigma_i \quad \text{and} \quad \prod_j x_j = \text{id} \Rightarrow \prod_j f_i(x)_j = \text{id}.$$ 

Finally concatenating all $f_i$ in order gives the stated function $f$.

We remark that Theorem 3.3.19 has an alternate, more involved proof which was discovered first and is given in Appendix A. This alternate proof differs in two respects. First, it holds only for $t \equiv 2 \pmod{4}$. Second and more notably, for certain choices of $\alpha$ and $\beta$ it gives a function $f$ with shorter output length $O(t)$ rather than $O(t^2)$. Specifically this holds whenever $\beta$ is a $k$-cycle and $\alpha$ is a product of $\Omega(k)$ disjoint transpositions. This does not improve the overall result in this section, but it may be useful in future work.
3.3.5.2 Hardness for a single element

In this section we show that the \((1 \ 2)(3 \ 4)\)-product problem is L-complete.

**Theorem 3.3.21.** If for sufficiently large \(t\) there is a circuit of depth \(O(\log t)\) that decides the \((1 \ 2)(3 \ 4)\)-product problem over \((A_t)^t\), then \(NC^1 = L\).

We use the following theorem which is proved afterwards and says that deciding if an input vector has product \(= \text{id}\) is L-complete.

**Theorem 3.3.22.** If for sufficiently large \(t\) there is a circuit of depth \(O(\log t)\) that decides if its input in \((A_t)^t\) has product \(= \text{id}\), then \(NC^1 = L\).

To prove Theorem 3.3.21 from Theorem 3.3.22, we show how to construct a set of \(t^{O(1)}\) vectors from an input vector \(x \in (A_t)^t\) such that, if \(x\) has product \(= \text{id}\) then they all do, and otherwise some vector has product \(= (1 \ 2)(3 \ 4)\). Then we apply the circuit deciding the \((1 \ 2)(3 \ 4)\)-product problem to each vector, and a depth-\(O(\log t)\) OR tree to the outputs. The vectors are constructed using commutation and conjugation via Lemma 3.3.20.

**Proof of Theorem 3.3.21.** Assume that there is a circuit \(C\) of depth \(O(\log t)\) that decides the \((1 \ 2)(3 \ 4)\)-product problem. We construct a circuit \(C'\) of depth \(O(\log t)\) that decides if its input has product \(= \text{id}\), which in combination with Theorem 3.3.22 proves the theorem.

Lemma 3.3.20 shows that for every \(\text{id} \neq \alpha \in A_t\), there exist \(\gamma_1, \gamma_2 \in A_t\) such that \(\alpha' := [[\alpha, \gamma_1], \gamma_2]\) is a double-transposition and each \(\gamma_i\) is either a double-transposition or a 3-cycle. We observe in the claim following this proof that for every double-transposition \(\alpha'\), there exists \(\gamma_3 \in A_t\) such that \(\gamma_3^{-1} \cdot \alpha' \cdot \gamma_3 = (1 \ 2)(3 \ 4)\) and \(\gamma_3\) permutes \(\leq 8\) points.

For any such choice of \(\gamma := (\gamma_1, \gamma_2, \gamma_3)\), let \(C_\gamma : (A_t)^t \to (A_t)^t\) denote a circuit of depth \(O(\log t)\) that satisfies

\[
\prod_i x_i = \alpha \implies \prod_i C_\gamma(x)_i = \gamma_3^{-1} \cdot [[\alpha, \gamma_1], \gamma_2] \cdot \gamma_3
\]

for every \(\alpha \in A_t\) and \(x \in (A_t)^t\). The crucial point is that if \(\prod_i x_i \neq \text{id}\) then there exists \(\gamma\) such that \(\prod_i C_\gamma(x)_i = (1 \ 2)(3 \ 4)\), and otherwise \(\prod_i C_\gamma(x)_i = \text{id}\) for every \(\gamma\).

Observe that the number of double-transpositions in \(A_t\) is \(\binom{t}{2} \cdot 3\), the number of 3-cycles is \(\binom{t}{3} \cdot 2\), and the number of permutations that permute \(\leq 8\) points is \(< \binom{t}{4} \cdot |A_8|\), all of which are \(t^{O(1)}\). Thus on input \(x\), \(C'\) checks in depth \(O(\log t)\) if any of these \(t^{O(1)}\) choices of \(\gamma = (\gamma_1, \gamma_2, \gamma_3)\) satisfies \(C(C_\gamma(x)) = 1\). \(\square\)
Claim. Let \( t \geq 8 \). For every double-transposition \( \alpha \in A_t \), there exists \( \gamma \in A_t \) such that 
\[
\gamma^{-1} \alpha \gamma = (1 \ 2)(3 \ 4)
\]
and \( \gamma \) permutes \( \leq 8 \) points.

Proof. Denote \( \alpha = (a \ b)(c \ d) \). Any injective function \( \phi : \{1, 2, 3, 4\} \cup \{a, b, c, d\} \to [8] \) maps \( \alpha \) and \((1 \ 2)(3 \ 4)\) to two double-transpositions in \( A_8 \). Since the latter are conjugate, and since the \( \gamma \in A_8 \) that “witnesses” this conjugacy necessarily permutes \( \leq 8 \) points, applying \( \phi^{-1} \) (suitably defined) to \( \gamma \) yields an element in \( A_t \) that permutes \( \leq 8 \) points and witnesses the conjugacy of \( \alpha \) and \((1 \ 2)(3 \ 4)\).

3.3.5.3 Proof of Theorem 3.3.22

Recall from above the following encoding of branching programs by permutations. The proof of this theorem is implicit in [CM87, Prop. 1].

**Theorem 3.3.18** ([CM87]). There exists \( t = O(s) \) and a circuit \( C \) of depth \( O(\log s) \) such that, on input \((B, x)\) where \( B \) is a branching program of size \( s \geq |x| \), \( C \) outputs a permutation \( \sigma \in S_t \) such that \( B \) accepts \( x \) iff 1 and \( t \) are in the same cycle in \( \sigma \)'s disjoint cycle representation.

Next we use this to show that the problem of deciding if a vector over \( S_t \) has product = \( \text{id} \) is L-complete. This proof is due to Eric Allender and V. Arvind (personal communication), and we include it with their permission.

**Theorem 3.3.23.** If for sufficiently large \( t \) there is a circuit of depth \( O(\log t) \) that decides if its input in \((S_t)^t\) has product = \( \text{id} \), then \( \text{NC}^1 = \text{L} \).

Proof. Let a string \( x \) and a branching program \( B \) of size \( s = \text{poly}(|x|) \) be given. Let \( t = O(s) \) be as in Theorem 3.3.18.

We first construct \( t \) vectors in \((S_t)^t\) such that \( B \) accepts \( x \) iff the product of some vector is a permutation that maps \( 1 \mapsto t \). Let \( \sigma \in S_t \) be given by Theorem 3.3.18. Notice that 1 and \( t \) are in the same cycle in \( \sigma \)'s disjoint cycle representation iff \( \exists k \leq t \) such that \( \sigma^k \) maps \( 1 \mapsto t \). Thus for \( k = 1, \ldots, t \) we construct the \( k \)th vector to have product \( \sigma^k \), by concatenating \( k \) copies of \( \sigma \) and \( t - k \) copies of \( \text{id} \). (Up to now this construction appears in [CM87].)

Next we transform \( z \in (S_t)^t \) to \( z' \in (S_{t+1})^{2t+2} \) satisfying

\[
\prod_i z_i \text{ maps } 1 \mapsto t \iff \prod_i z'_i = \text{id}. \tag{3.10}
\]

This is done via the map

\[
z' := (z, (t \ t + 1), z^{-1}, (1 \ t + 1))
\]
where \( z^{-1} := (z_1^{-1}, \ldots, z_{t+1}^{-1}) \in (S_t)^t \) and we embed \( S_t \) into \( S_{t+1} \) in the canonical way. This is computable in depth \( O(\log t) \).

To see that (3.10) holds, denote \( \pi := \prod_i z_i \) and \( \pi' := \prod_i z_i' \). If \( \pi(1) \neq t \), then \( \pi'(1) = t + 1 \) and so \( \pi' \neq \text{id} \). On the other hand if \( \pi(1) = t \), then it can be checked that \( \pi'(1) = 1 \) and \( \pi'(t + 1) = t + 1 \), and it is clear that \( \pi'(j) = j \) for \( 1 < j < t + 1 \) since \( \pi(j) \) is not touched by \( (t \ t + 1) \) and \( j \) is not touched by \( (1 \ t + 1) \).

Thus we have constructed \( t \) vectors in \((S_{t+1})^{2t+2}\) such that \( B \) accepts \( x \) iff some vector has product = \( \text{id} \). We can reduce the vectors’ length to \( t+1 \) in depth \( O(\log t) \) by multiplying adjacent permutations. Finally, applying the circuit in the assumption of the theorem and an OR-tree yields a circuit of depth \( O(\log t) \) that decides if \( B \) accepts \( x \). \( \square \)

To conclude the proof of Theorem 3.3.22, we observe that there is an embedding \( M : S_t \rightarrow A_{t+2} \) computable in \( \text{NC}^4 \) that preserves the identity product. \( M \) is defined by \( M(\alpha) := \alpha \) if \( \alpha \) is even and \( M(\alpha) := \alpha \cdot (t + 1 \ t + 2) \) if \( \alpha \) is odd. It can be checked that this is a homomorphism, and thus \( \prod_i x_i = \text{id} \Leftrightarrow \prod_i M(x_i) = \text{id} \). Computing \( M \) requires deciding if \( \alpha \in S_t \) is odd or even, which can be done in depth \( O(\log t) \) by checking if there are an odd or even number of pairs \( i < j \leq t \) such that \( \alpha(i) > j \).

### 3.4 Proofs of Corollary 3.1.6 and Theorem 3.1.8

In this section we prove Corollary 3.1.6 and Theorem 3.1.8, restated for convenience.

**Corollary 3.4.1.** There is a single efficient compiler \( \text{Comp} \), outputting a circuit \( \hat{C} \) of size \( |\hat{C}| = \text{poly}(|C|) \), that is an \( (L, \epsilon) \)-leakage secure compiler for each of the following.

1. \( L = \) number-in-hand protocols with \( s \) parties communicating and outputting \( \leq \delta \cdot |\hat{C}|^{1/3} \) bits, for a fixed \( \delta > 0 \) and a fixed partition of \( \hat{C} \) into \( s = O(1) \) sets; \( \epsilon = 2^{-\Omega(|\hat{C}|^{1/3})} \).
2. If \( \text{NC}^4 \neq L \) then for every \( k \) and infinitely many \( |C| \), \( L = \text{NC}^4 \) circuits with depth \( \leq k \log |\hat{C}| \) and \( k \log |\hat{C}| \) bits of output; \( \epsilon = |\hat{C}|^{-k} \).
3. If \( \text{TC}^0 \neq \text{NC}^1 \) then for every \( k \) and infinitely many \( |C| \), \( L = \text{TC}^0 \) circuits with size \( \leq |\hat{C}|^k \) and \( k \log |\hat{C}| \) bits of output; \( \epsilon = |\hat{C}|^{-k} \).
4. \( L = \text{AC}^0 \) circuits with depth \( \leq d \), size \( \leq |\hat{C}|^{O_d(\log |\hat{C}|)} \), an additional \( O_d(\log^2 |\hat{C}|) \) arbitrary symmetric gates, and \( |\hat{C}|^{0.01} \) bits of output; \( \epsilon = |\hat{C}|^{-\Omega_d(\log |\hat{C}|)} \).
5. \( \mathcal{L} = \text{AC}^0 \) circuits with depth \( \leq d \), size \( \leq 2^{O_d(|\mathcal{C}|(1-\delta)/3d)} \), and \(|\mathcal{C}|^{d/3}\) bits of output, for any \( \delta < 1; \epsilon = 2^{-O_d(|\mathcal{C}|(1-\delta)/3d)} \).

Proof. As mentioned in §3.1, items 2-5 follow in a straightforward manner from Theorem 3.1.4 and the results of §3.3. (For all five items, we choose \( t = |C| \) in Theorem 3.1.4.) Next we prove item 1.

The key is to show that the local extension of any number-in-hand (NIH) protocol, under any partition satisfying a certain restriction discussed below, is computable by a number-on-forehead (NOF) protocol under a corresponding partition. This allows us to show that any NIH protocol breaking the security of \( \hat{C} \), in combination with the local reduction from Theorem 3.1.4, gives an NOF protocol that is not fooled by \((A_{t})^t\) which contradicts item 1 of Theorem 3.1.5. We now give the details.

Let \( F = \{ f : G^t \to \{0,1\}^{\hat{C}} \} \) denote the set of possible reductions from the proof of Theorem 3.1.4. Let \( P \) be any \( s \)-party NOF partition of strings in \( G^t \). We assign to each wire \( j \in \hat{C} \) a set \( S_j \subseteq [s] \) of NOF players with the following property: for every \( f \in F \) and \( x \in G^t \), \( j \)'s value in \( f(x) \) depends only on elements of \( x \) that are on the foreheads of players in \( S_j \). Then if a given NIH partition \( P' \) of \( \hat{C} \) satisfies \( \bigcup_{j \in P'_i} S_j \subseteq [s] \) for every set \( P'_i \in P' \), we say that \( P' \) is simulatable. The key is that for any NIH protocol \( \ell \) under a simulatable partition \( P' \) and any \( f \in F \), \( \ell(f(\cdot)) \) is computable by an \( s \)-party NOF protocol under partition \( P \) with the same amount of communication. Indeed, player \( i \)'s communication in the NIH protocol can be simulated in the NOF protocol by any player in \([s] \setminus \bigcup_{j \in P'_i} S_j \).

To construct our simulatable NIH partition, we note that the set of reductions \( F \) given by the proof of Theorem 3.1.4 is the union of two sets \( F_1 \) and \( F_2 \), where \( F_1 \) contains the reductions arising from Claim \( i \). \( F_1 \) and \( F_2 \) each contain \( O(1) \)-local reductions as previously discussed, but in fact they satisfy the stronger property that every bit \( j \) in the output depends on the same \( O(1) \) inputs for every function in the set. Formally, the following holds \( \forall j \leq |\hat{C}| \).

\[
|\{i \leq t \mid \exists f \in F_1 \text{ whose } j\text{th output depends on its } i\text{th input}\}| \leq 1
\]

\[
|\{i \leq t \mid \exists f \in F_2 \text{ whose } j\text{th output depends on its } i\text{th input}\}| \leq 4
\]

Now let \( P \) be the partition of strings in \((A_{t})^t\) onto \( s = 6 \) foreheads given by Theorem 3.3.6. We define a simulatable partition \( P' \) of strings in \( \{0,1\}^{\hat{C}} \) into 6 hands as follows. For each wire \( j \leq |\hat{C}| \), we assign \( j \) to hand \( i \) for some \( i \in [6] \) such that for every \( f \in F_1 \cup F_2 \), the \( j \)th output bit of \( f \) does not depend on any input element on forehead \( i \) in \( P \). This is possible because
each output bit depends on \( \leq 4 + 1 = 5 \) input elements and thus \( \leq 5 \) foreheads of \( P \), and \( P' \) is simulatable because we have \( S_j \subsetneq [6] \) for every \( j \).

To complete the proof, recall that Theorem 3.1.4 shows that for every \( k \in \{0, 1\}^n \), if there is a function \( \ell \) on domain \( \{0, 1\}^{\hat{C}|} \) and an \( x \in \{0, 1\}^n \) such that

\[
\Delta(\ell(\hat{W}_x), \ell(S(C, x, \hat{C}(x)))) \geq \epsilon \cdot t \cdot |C| \tag{3.11}
\]

for \( \hat{C} \leftarrow \text{Comp}(C, k) \), then there exists \( \alpha \in G \) and a function \( f \in F_1 \cup F_2 \) such that

\[
\Delta(\ell(f(D_\alpha)), \ell(f(U_{G^t}))) \geq \epsilon. \tag{3.12}
\]

If (3.11) holds for a NIH protocol \( \ell \) under \( P' \), then (3.12) contradicts item 1 of Theorem 3.1.5. \( \Box \)

The previous proof constructed a simulatable NIH partition \( P \) of \( \hat{C} \) into 6 sets each of size \( O(|\hat{C}|) = O(t^2 \cdot |C| \cdot \log |G|) \). To analyze the security of our compiler in the only computation leaks (OCL) model, we now show that this partition can be refined into a simulatable, topologically-ordered partition with \( O(t \cdot |C|) \) sets each of size \( O(t \cdot \log |G|) \). (Recall from §3.1.2 that \( P \) is topologically-ordered if for each \( P_i \in P \) and each wire \( j \in P_i \), \( j \)'s value in \( \hat{C} \)'s computation depends only on wires \( j' \) such that \( j' \in P_{i'} \) for some \( i' \leq i \).) In the following, the canonical partition of \( x \in G^t \) for \( s \)-party NOF protocols is the one in which player \( i \)'s forehead contains \( x_i, x_{i+s}, \ldots, x_{i+t-s} \).

**Theorem 3.1.8.** Assume that 8-party NOF protocols communicating \( \leq c \) bits are \( \epsilon \)-fooled by \( G^t \) under the canonical partition of \( x \in G^t \).

Then for each \( C \) and \( k \) there is a topologically-ordered partition \( P \) on \( \hat{C} := \text{Comp}(C, k) \) containing \( O(t \cdot |C|) \) sets each of size \( O(t \cdot \log |G|) \), such that \( \text{Comp} \) from Theorem 3.1.4 is an \( (\mathcal{L}, \epsilon \cdot t \cdot |C|) \)-leakage secure compiler for \( \mathcal{L} = \text{all OCL leakage functions that output} \leq \delta \cdot c/t \) bits per set in \( P \), where \( \delta \) is a constant that depends only on the maximum fanout of \( C \).

**Proof.** In the OCL model, the compiler specifies a topologically-ordered partition \( P \) on the wires of the compiled circuit \( \hat{C} \), and the adversary adaptively chooses leakage functions to be applied to the wires of each set in \( P \). We observe that any such adversary defines a number-in-hand (NIH) communication protocol on \( \hat{C}(x) \)'s wires under \( P \). Thus following the proof above, any OCL adversary under a simulatable partition \( P \) that breaks the security of \( \hat{C} \) contradicts item 1 of Theorem 3.1.5. (For this theorem we consider 8-party protocols rather than 6-party solely because it makes the partition cleaner to describe.)
Recall the two classes of reduction functions \( f : G^t \rightarrow \{0, 1\}^{\widehat{C}} \) in Theorem 3.1.4. In the first class, \( f(x) \) plugs \( x \) into the bundle \( L \) in some multiplication tree of some random gadget, and reconstructs the other wires in the tree via the method described in Lemma 3.2.1; all wires outside this tree (and the gadget) are fixed in \( f \)'s output by the hybrid argument.

In the second class, \( f(x) \) plugs \( x \) into a bundle at the output of some random gadget (thus also at the input of some NAND gadget). It then computes honestly the wires in the NAND gadget, reconstructs the random gadget that outputs \( x \), and reconstructs the random gadgets that take the NAND gadget’s output as input. All other wires are fixed in \( f \)'s output by the hybrid argument.

We now describe the partition \( P \) and the sets \( S_w \subseteq [8] \) for each wire \( w \in \widehat{C} \). (Recall that for \( P \) to be simulatable, each \( S_w \) must have the property that \( w \)'s value in the output of any reduction \( f(x) \) depends only on the foreheads of players in \( S_w \) under the canonical partition of \( x \).) \( P \) first splits \( \widehat{C} \) into \( O(|C|) \) sets corresponding to the gadgets, and then refines each gadget’s set as described below.

**Partition for NAND.** First let \( w \) be a wire in a NAND gadget \( N \). If \( w \) is \( N \)'s \( i \)th input wire \((1 \leq i \leq t)\), then \( S_w = \{i \mod 8\} \). This is because for each \( f \in F \), \( w \)'s value in \( f(x) \)'s output is either fixed by the hybrid argument or is equal to \( x_i \) which is on the forehead of player \( i \mod 8 \).

If \( w \) is a wire at the output of an inversion gate whose input is \( N \)'s \( i \)th input wire, then we again have \( S_w = \{i \mod 8\} \).

If \( w \) is a wire at the output of a multiplication gate in \( N \) (including \( N \)'s output wires), we have either \( S_w = \{1, 2, 3, 4\} \) or \( S_w = \{5, 6, 7, 8\} \). This is because each multiplication gate depends on \( \leq 4 \) consecutive elements from \( N \)'s input.

Thus we partition each NAND gadget into two sets of equal size \( O(t \log |G|) \): the wires \( w \) such that \( S_w \subseteq \{1, 2, 3, 4\} \), and the wires \( w \) such that \( S_w \subseteq \{5, 6, 7, 8\} \). Note that so far \( P \) is topologically ordered and simulatable.

**Partition for RANDOM.** Now let \( w \) be a wire in a random gadget \( D \). If \( w \) is \( D \)'s \( i \)th output wire then we have \( S_w = \{i \mod 8\} \) for the same reason as above: \( w \)'s value is either fixed by the hybrid argument or is equal to \( x_i \).

If \( w \) is one of \( D \)'s input wires then by construction it is an output wire of some NAND gadget, and so we already have \( S_w = \{1, 2, 3, 4\} \) or \( S_w = \{5, 6, 7, 8\} \). Recall that each input wire of \( D \) becomes the middle input to a \((2t + 1)\)-wise multiplication tree, in which the leftmost \( t \) inputs
are denoted $L$ and the rightmost $t$ inputs are denoted $R$.

If $w$ is the $i$th wire ($1 \leq i \leq t$) in a bundle $L$ of some tree, then we have $S_w = \{i \text{ mod } 8\}$ because again $w$’s value is either fixed by the hybrid argument or is equal to $x_i$.

If $w$ is the $i$th wire in a bundle $R$ of some tree, then by the reconstruction procedure in Lemma 3.2.1 $w$ depends only on its “mirror image” in $L$, i.e. the $(t - i + 1)$th wire of $L$; thus we have $S_w = \{t - i + 1 \text{ mod } 8\}$. There are two exceptions to this. First, the first wire of $R$ also depends on the middle input wire of the tree, i.e. one of $D$’s input wires (recall that this dependence is in the reconstruction procedure, not in the gadget’s computation itself); thus for this wire we have either $S_w = \{1, 2, 3, 4, 8\}$ or $S_w = \{5, 6, 7, 8\}$. Second, the last wire of $R$ also depends on the output wire of this tree, i.e. one of $D$’s output wires; thus for this wire we have $S_w = \{1, m\}$ for some $m \in [8]$.

We now illustrate the sets $S_w$ for the $2t + 1$ input wires of a multiplication tree in $D$. Either they are

$$
\begin{array}{c}
\underbrace{\{1\} \cdots \{8\}}_{L} \cdots \underbrace{\{1\} \cdots \{8\}}_{L} \quad \underbrace{\{1, 2, 3, 4\}}_{R} \cdots \underbrace{\{7\} \cdots \{8\}}_{R} \cdots \underbrace{\{2\} \{1, m\}}_{R}
\end{array}
$$

for some $m \in [8]$, or else they are

$$
\begin{array}{c}
\underbrace{\{1\} \cdots \{8\}}_{L} \cdots \underbrace{\{1\} \cdots \{8\}}_{L} \quad \underbrace{\{5, 6, 7, 8\}}_{R} \cdots \underbrace{\{7\} \cdots \{8\}}_{R} \cdots \underbrace{\{2\} \{1, m\}}_{R}
\end{array}
$$

$P$ partitions each such set of input wires as follows, which induces a partition on the whole tree. Essentially, $P$ works from the outside towards the middle, grouping blocks of 4 consecutive wires from each side. So, the first set contains wires $1, \ldots, 4$ of $L$ and wires $(t - 3), \ldots, t$ of $R$, the next set contains wires $5, \ldots, 8$ of $L$ and wires $(t - 7), \ldots, (t - 4)$ of $R$, and so on. The middle $8 + 1 + 8 = 17$ wires are handled differently, but note that for each set so far we have $\bigcup_w S_w = \{5, 6, 7, 8\}$ or $\bigcup_w S_w = \{1, 2, 3, 4, m\}$ for some $m \in [8]$.

We partition the middle 17 wires into two sets as follows. If the middle wire has $S_w = \{5, 6, 7, 8\}$ then we partition as the other wires: the outermost 8 wires form one set (with $\bigcup_w S_w = \{1, 2, 3, 4\}$), and the innermost 9 form the other (with $\bigcup_w S_w = \{5, 6, 7, 8\}$). If instead the middle wire has $S_w = \{1, 2, 3, 4\}$, then we make one set from the outermost 10 wires (with $\bigcup_w S_w = \{1, 2, 3, 4, 5\}$), and the other from the innermost 7 wires (with $\bigcup_w S_w = \{1, 2, 3, 4, 6, 7, 8\}$).

The partition on the input wires induces a partition on the whole tree in the natural way: each internal wire, which is the output of some multiplication gate, is assigned to the set containing the input wire on which it depends. This preserves the value of each $\bigcup_w S_w$ listed above, and
thus the partition is simulatable. It is also topologically ordered, as the sets in the partition form “concentric subtrees” that can be evaluated from the inside out.

For each tree, we have created a partition into $O(t)$ sets each of size $O(\log |G|)$. (Here the hidden constants depend on $s = O(1)$.) Since there are $t$ trees in a random gadget, naively this partitions the whole gadget into $O(t^2)$ sets each of size $O(\log |G|)$. However, by combining the sets from each tree that are at the same depth in the concentric subtrees, we can instead get a partition into $O(t)$ sets each of size $O(t \log |G|)$. This partition remains simulatable and topologically ordered.

Overall, we have $O(t \cdot |C|)$ sets each of size $O(t \log |G|)$ as promised.

**From NIH to NOF.** Assume that, for some circuit $C$ and string $k$, there is an OCL adversary $\ell$ under partition $P$ and an input $x$ such that $\Delta(\ell(\hat{W}_x), \ell(S(C, x, \hat{C}(x)))) \geq \epsilon \cdot t \cdot |C|$, where $\hat{C} \leftarrow \text{Comp}(C, k)$ and $S$ is the simulator from Theorem 3.1.4. Then by Theorem 3.1.4 there is some $\alpha \in G$ and $f \in F$ such that $\Delta(\ell(f(D_\alpha)), \ell(f(U_G^t))) \geq \epsilon$. By construction, the function $\ell(f(x))$ can be computed by an $s$-party NOF protocol under the canonical partition of $x \in G^t$, because $\ell$ corresponds to an NIH protocol and $P$ is simulatable. The only remaining question is how much communication is required in the NOF protocol.

Let $b \in \mathbb{N}$ be a bound on the output length of each leakage function chosen by the OCL adversary, i.e. a bound on the amount of communication by each player in the NIH protocol. Let $d$ denote the maximum fanout of any gate in $C$. The key point is that for each $f \in F$, all wire values in $f(x) \in \{0, 1\}^{\hat{|C|}}$ are fixed by the hybrid argument (independent of $x$) except for in at most $d + O(1)$ gadgets, and thus in at most $O(td)$ sets in $P$. Since no communication is needed to simulate the NIH players whose inputs are fixed, the NOF protocol communicates $O(tdb)$ bits in total. Thus if $b = \delta c/t$ for sufficiently small $\delta$ that depends only on $d$, this contradicts the assumption of the theorem.

### 3.5 Multi-query security via secure hardware

In this section we consider the extension of our construction to the setting where the adversary can make multiple, adaptive queries. We follow the approach of \cite{FRR+10}, using a simple secure hardware component. As mentioned in §3.1.1, in our setting this component has no input and outputs a sample from the distribution $D_{id}$.

We generalize Definition 3.1.1 to the multi-query setting following \cite[Definition 1]{FRR+10},
beginning with an overview. As before, the adversary $A$ is restricted to choosing leakage functions from some class $\mathcal{L}$ and remains otherwise computationally unbounded. $A$ makes $q$ queries to the circuit $\hat{C}$, denoted $(x_i, \ell_i) \in \{0,1\}^n \times \mathcal{L}$ for $i \leq q$, and in response to the $i$th query $A$ receives $(\hat{C}(x_i), \ell_i(\hat{W}_i))$ where $\hat{W}_i$ denotes the wires of $\hat{C}(x_i)$. $A$ chooses its queries adaptively, meaning that each query can depend on all responses seen so far.

On input $(C, k)$, the compiler from Theorem 3.1.4 implicitly computes a string $\hat{k}_0 \in (G^t)^n$ that encodes $k$. In the multi-query setting, this encoding must be “refreshed” between queries, as otherwise $A$ can learn, say, the first bit of $k$ after $O(t)$ queries. To accomplish this, the compiler below outputs a circuit $\hat{C} : \{0,1\}^n \times (G^t)^n \rightarrow \{0,1\}^n \times (G^t)^n$ as well as an initial encoding $\hat{k}_0$. Then, the second output $\hat{k}_i$ of $\hat{C}(x_i, \hat{k}_{i-1})$ is used as the second input to $\hat{C}$ in the $(i+1)$th query. (This corresponds to the notion of a stateful circuit in $[\text{FRR}^+10]$.) Crucially $A$ does not directly obtain any $\hat{k}_i$, but the leakage functions operate on these values as they are carried on wires of $\hat{C}$.

For an adversary $A$ interacting with such a circuit $\hat{C}$ on initial encoding $\hat{k}_0$, we let $(x_1, \ell_1) = A(\hat{C})$ be the first query, and then inductively define

\[
(y_i, \hat{k}_i) := \hat{C}(x_i, \hat{k}_{i-1})
\]

\[
(x_{i+1}, \ell_{i+1}) := A(\hat{C}, x_1, y_1, \ell_1(\hat{W}_1), \ldots, x_i, y_i, \ell_i(\hat{W}_i)).
\]  

(3.13)

We note two final differences from the single-query setting. First, the circuit $\hat{C}$ is randomized, which means that it contains gates whose output comes from some distribution rather than being deterministically fixed by the input. (In our construction, the only randomized gates will be the secure hardware components which output a sample from $D_{\text{id}}$.) Second, the simulator $S$ is stateful, which means that between subsequent evaluations it maintains some state. (In our construction the state will be an element of $(G^t)^n$.)

**Definition 3.5.1.** Let $\text{Comp}(\cdot, \cdot)$ be a randomized algorithm that takes as input a circuit $C : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$ and a string $k \in \{0,1\}^n$. For a set of functions $\mathcal{L}$, $\text{Comp}$ is a $q$-query $(\mathcal{L}, \epsilon)$-leakage-secure compiler if the following properties hold.

1. (Structure.) For every $C$ and $k$, $\text{Comp}(C, k)$ outputs a string $\hat{k}_0 \in (G^t)^n$, and a randomized circuit $\hat{C} : \{0,1\}^n \times (G^t)^n \rightarrow \{0,1\}^n \times (G^t)^n$ which is completely determined by $C$.

2. (Correctness.) For every $A$ as above, every $C, k$, and every $i \leq q$: $y_i = C(x_i, k)$ with probability 1.
3. (Security.) There is a randomized polynomial-time stateful algorithm $S$ such that the following holds for every $C,k$ and every $A$ as above. Let $D_{\text{real}}$ denote the distribution $(\ell_1(\hat{W}_1), \ldots, \ell_q(\hat{W}_q))$, and let $D_{\text{sim}}$ denote the corresponding distribution
\[
(\ell_1(S(C, x_1, y_1)), \ldots, \ell_q(S(C, x_q, y_q)))
\]
when each $\hat{W}_j$ in (3.13) ($j \leq i$) is replaced with $S(C, x_j, y_j)$. Then, $\Delta(D_{\text{real}}, D_{\text{sim}}) \leq \epsilon$.

The construction. Recall from §3.1.1 that a $D_{\text{id}}$-gate is a gate with no input that on each execution of the circuit outputs a string of length $2t$ sampled from $D_{\text{id}}$, and any circuit that contains one or more $D_{\text{id}}$-gates is a $D_{\text{id}}$-circuit.

Then, $\text{Comp}$ outputs a $D_{\text{id}}$-circuit $\hat{C} : \{0, 1\}^n \times (G^t)^n \rightarrow \{0, 1\}^n \times (G^t)^n$ as follows. $\hat{C}$ is identical to the construction in §3.2 with two exceptions. First, each pair $(R(i), L(i+1))$ in each RANDOM gadget is computed by a $D_{\text{id}}$-gate. Second, $\hat{C}$ computes its second output $\hat{k}_i$ by applying a RANDOM gadget to each bundle of its second input $\hat{k}_{i-1}$.

The simulator $S$ then operates as follows. For the first query, $S$ chooses $\hat{k}_0, \hat{k}_1 \in (G^t)^n$ uniformly at random, and produces wire values for $\hat{C}(x_1, \hat{k}_0)$ conditioned on output $(y_1, \hat{k}_1)$ as described in Theorem 3.1.4. Between queries $i$ and $i + 1$, $S$ stores the value $\hat{k}_i$, and for the $(i + 1)$th query it chooses $\hat{k}_{i+1}$ uniformly at random and proceeds in the same manner.

Proving the security of this construction requires a slightly stronger property of the group encoding than what is given by Definition 3.1.2. Namely, it requires that the leakage class $L$ cannot distinguish the distributions $D_\alpha$ and $U_{G^t}$ even with two adaptive queries. The need for this is due to the fact that each $\hat{k}_i$ ($1 \leq i < q$) is given as input to two leakage functions: once when $\hat{k}_i$ is an output of $\hat{C}$ and once when it is an input. Formally, we require the following strengthening of Definition 3.1.2.

**Definition 3.5.2.** Let $G$ be a group and $t \in \mathbb{N}$. A set of functions $L$ is $2$-adaptive $\epsilon$-fooled by $G^t$ if for every $\alpha \in G$, every $\ell \in L$, and every function $A : \text{range}(\ell) \rightarrow L$, the following two distributions are $\epsilon$-close in statistical distance.

1. Sample $w \leftarrow D_\alpha$, compute $\ell' := A(\ell(w))$, and output $(\ell(w), \ell'(w))$.
2. Sample $w \leftarrow U_{G^t}$, compute $\ell' := A(\ell(w))$, and output $(\ell(w), \ell'(w))$.

With this stronger property, the following theorem can be proved by building on Theorem 3.1.4. Specifically, one first uses Theorem 3.1.4 to show that the security property of Definition 3.5.1 holds when the simulator chooses each $\hat{k}_i$ as in the real execution but the internal wires of
each evaluation $\tilde{C}(x_i, \tilde{k}_i)$ are reconstructed. Then, a hybrid argument over each bundle in each $\tilde{k}_i$ is used to show that the security property is satisfied even when $S$ chooses each $\tilde{k}_i$ uniformly at random. The proof is essentially identical to the proof of [FRR+10, Lemma 15], and we omit the details.

**Theorem 3.5.3.** Let $G$ be a group. For every polynomial-time computable function $t = t(n, |C|)$, there is a compiler $\text{Comp}$ for which the following holds.

1. For every $C : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ and $k \in \{0,1\}^n$, $\text{Comp}(C, k)$ runs in time $\text{poly}(|C|, t)$ and outputs a $D_{\text{ab}}$-circuit $\tilde{C}$ of size $O(t^2 \cdot |C|)$ and depth $O(t \cdot \text{depth}(C))$.

2. For every set of functions $L$, every $q \in \mathbb{N}$, and every $\epsilon > 0$, if the 4-local extension of $L$ is 2-adaptive $\epsilon$-fooled by $G^t$ then $\text{Comp}$ is a $q$-query $(L, \epsilon')$-leakage-secure compiler for $\epsilon' := (q + 1) \cdot \epsilon \cdot (n + t \cdot |C|)$.

Similarly to Corollary 3.1.6, Corollary 3.1.7 is derived as a result of Theorem 3.5.3 and the following subsection, by choosing $t = |C|$.

### 3.5.1 Adaptive compression bounds

In this section, we show that the function classes from §3.3 are 2-adaptive fooled by $(A_5)^t$. For the class of functions computable by $\text{AC}^0$ with symmetric gates, this requires asymptotically smaller output length.

#### 3.5.1.1 Multi-party protocols

**Theorem 3.5.4.** There is a partition of the inputs in $(A_5)^t$ into $k$ pieces such that the set $L$ of $k$-party number-on-forehead protocols communicating $c$ bits and outputting $\leq c$ bits is 2-adaptive $\epsilon$-fooled by $(A_5)^t$ for $\epsilon = 2^{\Omega(t/(k^2 \cdot 4^k))}$.  

**Proof.** The partition is the same as in Theorem 3.3.6. Assume that $L$ is not 2-adaptive $\epsilon$-fooled by $(A_5)^t$, and let $\alpha \in A_5$, $P' \in L$, and $A : \text{range}(P') \to L$ violate Definition 3.5.2.

Consider the following $k$-party protocol $P$ that communicates and outputs $2c$ bits. On input $x$, the parties first compute $P'(x)$ by communicating $c$ bits. Then, they each determine $P'' := A(P'(x))$ with no communication. Finally, they compute $P''(x)$ again communicating $c$ bits. By assumption we have $\Delta(P(D_{\alpha}), P(U_{(A_5)^t})) \geq \epsilon$, which contradicts Theorem 3.3.6. 

\qed
3.5.1.2 \( \text{TC}^0 \)

**Theorem 3.5.5.** If \( \text{TC}^0 \neq \text{NC}^1 \) then \( \forall k \) and infinitely many \( t \), the class \( \mathcal{L} \) of \( \text{TC}^0 \) circuits with size \( \leq t^k \) and output length \( k \log t \) is 2-adaptive \( t^{-k} \)-fooled by \((A_5)^t\).

**Proof.** Assume that there exists \( k \) such that for sufficiently large \( t \), \( \mathcal{L} \) is not 2-adaptive \( t^{-k} \)-fooled by \((A_5)^t\). Let \( \alpha \in A_5 \), \( \ell : (A_5)^t \rightarrow \{0,1\}^{k \log t} \), and \( A : \{0,1\}^{k \log t} \rightarrow \mathcal{L} \) be the choices that violate Definition 3.5.2. Let \( \ell_1', \dotsc, \ell_t' \in \mathcal{L} \) be all possible circuits that \( A \) could output.

Let \( \ell'' : (A_5)^t \rightarrow \{0,1\}^{2k \log t} \) be the following procedure. On input \( x \in (A_5)^t \): first compute \( \ell(x) \), then select \( \ell' := \ell_{\ell(x)}' \) (identifying \( \ell(x) \in \{0,1\}^{k \log t} \) with the natural number it represents), and finally output \( (\ell(x), \ell'(x)) \). Clearly \( \ell'' \) is computable by a \( \text{TC}^0 \) circuit of size \( t^{O(k)} \), and by assumption we have \( \Delta(\ell''(D_\alpha), \ell''(U(A_5)^t))) \geq t^{-k} \) which contradicts Theorem 3.3.14.

The proofs of the next two theorems are essentially identical to the preceding one, given the corresponding result in §3.3.

3.5.1.3 \( \text{NC}^1 \)

**Theorem 3.5.6.** If \( \text{NC}^1 \neq L \) then \( \forall k \) and infinitely many \( t \), the class \( \mathcal{L} \) of \( \text{NC}^1 \) circuits with depth \( \leq k \log t \) and output length \( k \log t \) is 2-adaptive \( t^{-k} \)-fooled by \((A_1)^t\).

3.5.1.4 \( \text{AC}^0 \)

**Theorem 3.5.7.** For every \( 0 < \delta < 1 \) and every integer \( d > 0 \), there is a constant \( \epsilon > 0 \) such that the following holds. Let \( \mathcal{L} \) be the class of unbounded-fanin circuits \( C : (A_5)^t \rightarrow \{0,1\}^{t^\delta} \) of depth \( \leq d \) and size \( \leq 2^{\epsilon t(1-\delta)/d} \). Then, \( \mathcal{L} \) is 2-adaptive \( 2^{-\epsilon t(1-\delta)/d} \)-fooled by \((A_5)^t\).

3.5.1.5 \( \text{AC}^0 \) with symmetric gates

Recall that in §3.3.3, it was shown that \((A_5)^t \ t^{-\Omega(\log t)}\text{-fools the class of unbounded-fan-in constant-depth circuits that contain } t^{O(\log t)} \text{ And/Or/Not gates and } O(\log^2 t) \text{ arbitrary symmetric gates and output } t^{0.1} \text{ bits.} \) To apply the technique from the two preceding proofs to this class, one needs to restrict the output length to \( O(1) \) to ensure that the circuit \( \ell'' \) still contains only \( O(\log^2 t) \) symmetric gates. However, by a more careful extension of Theorem 3.3.11 we can improve the output length to \( \Omega(\log^2 t) \). In the following, we focus mainly on the necessary changes to the proof of Theorem 3.3.11.
Theorem 3.5.8. For every $d$, there is an $\epsilon > 0$ such that the following holds for every $t$.

Let $\mathcal{L}$ be the set of functions $\ell : (A_5)^t \rightarrow \{0,1\}^{\epsilon \log^2 t}$ where each output bit of $\ell$ is computable by an unbounded-fan-in circuit of depth $\leq d$ that contains $\leq t^\epsilon \log t$ And/Or/Not gates and $\leq \epsilon \log^2 t$ arbitrary symmetric gates.

Then, $\mathcal{L}$ is 2-adaptive $t^{-\epsilon \log t} \cdot \text{fooled by } (A_5)^t$.

Proof. Assume that $\mathcal{L}$ is not 2-adaptive $t^{-\epsilon \log t} \cdot \text{fooled by } (A_5)^t$, and let $\alpha \in A_5$, $\ell_0 \in \mathcal{L}$, and $A : \{0,1\}^{\epsilon \log^2 t} \rightarrow \mathcal{L}$ be the choices that violate Definition 3.5.2. Let $\ell_1, \ldots, \ell_{\epsilon \log t} \in \mathcal{L}$ be all possible circuits that $A$ could output.

For an appropriate $n = \Omega(\sqrt{t}/\log t)$ and for each $i = 0, \ldots, t^\epsilon \log t$, let $\ell'_i : \{0,1\}^{n^{2-0.3 \log n}} \rightarrow \{0,1\}^{t^\epsilon \log^2 t}$ be the corresponding function given by Lemma 3.3.13. These functions have the property that the distribution $(\ell'_0(x), \ell'_{\ell_0(x)}(x))$ when $x \leftarrow PAP^{-1}(1)$ has statistical distance $\geq n^{-\epsilon \log n}$ from the corresponding distribution when $x \leftarrow PAP^{-1}(0)$, for $\epsilon' = \epsilon'(\epsilon) > 0$.

Let $R$ be the distribution on random restrictions $\rho$ given in the proof of Theorem 3.3.11. Say that $\rho$ is good if $PAP|_{\rho} = GIP$ (i.e. $PAP|_{\rho}$ has exactly 1 input restricted per bottom $\oplus$ gate) and for every $i \leq t^\epsilon \log t$ and $j \leq \epsilon \log^2 t$, $\ell'_{i,j}|_{\rho}$ is computable by a $(0.3 \log n)$-party protocol exchanging $\log^5 n$ bits (where $\ell'_{i,j}$ denotes the $j$th output bit of $\ell'_i$.) Because the number of $\ell'_{i,j}$ is $\epsilon \log^2 t \cdot (t^\epsilon \log t + 1)$ and because $t = n^{O(1)}$, when $\epsilon$ is sufficiently small we obtain

$$\Pr_{\rho \leftarrow R}[\rho \text{ is good}] \geq 1 - n^{\Omega(\log n)}$$

by combining [Vio07, Claim 11 & Lemma 12] with a union bound.

For any $\rho$ that is good, let $P_{\rho} : \{0,1\}^{n^{2-0.3 \log n}} \rightarrow \{0,1\}$ be the following $(0.3 \log n)$-party protocol exchanging $2\epsilon \log^2 t \cdot \log^5 n + 1 = \log^{O(1)} n$ bits. On input $y$, the parties first compute each of the $\epsilon \log^2 t$ output bits of $\ell'_{0,y}(y)$, exchanging a total of $\epsilon \log^2 t \cdot \log^5 n$ bits. Then, each party chooses $\ell'_{i} := \ell'_{i,y}(y)$ with no communication. Then, the parties compute each of the $\epsilon \log^2 t$ output bits of $\ell'_{i,y}(y)$ again exchanging $\epsilon \log^2 t \cdot \log^5 n$ bits. Finally, the parties use one additional bit of communication to output 1 iff $(\ell'_{0,y}(y), \ell'_{i,y}(y)) \in S$ for the appropriate set $S$ corresponding to (3.7) in Theorem 3.3.11.

The rest of the proof follows the same argument as Theorem 3.3.11.

3.6 Future directions

We conclude by mentioning two directions for future research. The first is to remove the use of secure hardware in the setting of an unbounded number of queries by the adversary. Secure-
hardware-free constructions have recently been given by Goldwasser and Rothblum [GR12] for OCL leakage functions, and by Rothblum [Rot12] for global AC⁰ leakage functions. (As mentioned previously, the latter comes at the expense of introducing a computational assumption.) One immediate obstacle to applying their techniques here is that they rely on the commutativity of addition over bits, while the security of our construction relies crucially on the fact that the group A_n is not commutative. Still it seems plausible that some analog of these constructions, and particularly the notion of “ciphertext banks” from [GR12], can be found in our setting.

The second is to prove the security of our construction in the OCL model. As discussed in §3.1.2, this would follow from new number-on-forehead communication lower bounds for computing iterated group products.
Bibliography


[FRR+10] Sebastian Faust, Tal Rabin, Leonid Reyzin, Eran Tromer, and Vinod Vaikuntanathan. Protecting circuits from leakage: the computationally-bounded and


Appendix A

Alternate proof of Theorem 3.3.19

Here we give an alternate proof of Theorem 3.3.19, restricted to the case \( t \equiv 2 \pmod{4} \).

**Theorem A.0.1.** Let \( t \equiv 2 \pmod{4} \) and let \( \text{id} \neq \alpha, \beta \in A_t \). Then for all \( m \) there is a 1-local function \( f : (A_t)^m \to (A_t)^{O(tm)} \) computable in depth \( O(\log t) \) that reduces the \( \alpha \)-product-problem to the \( \beta \)-product-problem, i.e. that satisfies \( \forall x = (x_1, \ldots, x_t) \in (A_t)^t \):

\[
\prod_i x_i = \alpha \Rightarrow \prod_i f(x)_i = \beta \quad \text{and} \quad \prod_i x_i = \text{id} \Rightarrow \prod_i f(x)_i = \text{id}.
\]

**Proof overview.** As in §3.3.5.1 we use conjugation and commutation to construct a locally computable, identity-preserving map from \( \alpha \) to \( \beta \). The construction has three main steps.

1. **Reduce to a product of two disjoint transpositions.** (Lemma 3.3.20)

   We first convert any \( \alpha \neq \text{id} \) into a product of two disjoint transpositions. For example if \( \alpha \)'s disjoint cycle representation contains a 4-cycle \( (a \ b \ c \ d) \), then commutating with \( \gamma := (a \ b)(c \ d) \) yields \( [\alpha, \gamma] = (a \ c)(b \ d) \).

2. **Grow the number of transpositions.** (Lemma A.0.4)

   From a product of disjoint transpositions, we use one commutation to double the number of transpositions, and \( \log k \) commutations to convert from 2 to \( 2^k \) disjoint transpositions. For example if \( \alpha = (a \ b)(c \ d) \), then commutating with \( \gamma := (a \ e)(b \ f)(c \ g)(d \ h) \) yields \( [\alpha, \gamma] = (a \ b)(c \ d)(e \ f)(g \ h) \).

3. **Combine transpositions into cycles.** (Lemmas A.0.5-A.0.6)

   We finally convert products of disjoint transpositions into longer cycles. For example if \( \alpha = (a_1 \ b_1) \cdots (a_k \ b_k) \), then commutating with \( \gamma := (a_1 \ b_1 \ a_2 \ b_2 \cdots a_k \ b_k \ c) \) yields the \( (2k + 1) \)-cycle \( [\alpha, \gamma] = (a_1 \cdots a_k \ b_k \cdots b_1 \ c) \).

In total we use \( \leq \log t + O(1) \) commutations. This turns out to be tight for certain starting and target permutations as shown in the following theorem, and thus for these permutations a map with smaller output length requires different techniques. (The theorem also holds if both commutations and conjugations are allowed.)
Theorem A.0.2. There exist \( \alpha, \beta \in A_t \) such that \( \alpha \) cannot be converted to \( \beta \) with fewer than \( \log(t) - 1 \) commutations. That is, for every \( \ell < \log(t) - 1 \) and every sequence \( \gamma_1, \ldots, \gamma_\ell \in A_t \), \( \beta \neq [[ \cdots [[\alpha, \gamma_1], \gamma_2], \cdots], \gamma_\ell] \).

Proof. For \( \alpha \in A_t \), let \( M(\alpha) := |\{ i \in [t] \mid \alpha(i) \neq i \}| \) denote the number of points moved (i.e. not fixed) by \( \alpha \). We show that \( M([\alpha, \gamma]) \leq 2 \cdot M(\alpha) \) for every \( \alpha \) and \( \gamma \), i.e. the number of points moved by \([\alpha, \gamma]\) is at most twice the number of points moved by \( \alpha \). This implies the theorem by choosing \( \alpha = (1 \ 2 \ 3) \) and \( \beta = (1 \ 2 \ \cdots \ t) \), because \( M(\alpha) = 3 \) and \( M(\beta) = t \).

Pick \( \gamma \in A_t \). Observe that \( \gamma \alpha^{-1} \gamma^{-1} \) has the same cycle type as \( \alpha \) because it is conjugate to \( \alpha \) in \( S_t \). This implies \( M(\gamma \alpha^{-1} \gamma^{-1}) = M(\alpha) \), and therefore \( [\alpha, \gamma] = \alpha \gamma \alpha^{-1} \gamma^{-1} \) moves at most \( 2 \cdot M(\alpha) \) points because any such point must be moved by either \( \alpha \) or \( \gamma \alpha^{-1} \gamma^{-1} \). \( \square \)

Proof details. The function \( f \) is constructed by concatenating compositions of functions from the following two families, where the compositions are given by Lemma A.0.3.

\[
\text{Conj} = \{ \text{Conj}_\gamma(\alpha) := \gamma^{-1} \alpha \gamma \mid \gamma \in A_t \} \quad \text{Comm} = \{ \text{Comm}_\gamma(\alpha) := \alpha \gamma \alpha^{-1} \gamma^{-1} \mid \gamma \in A_t \}
\]

Lemma A.0.3. Let \( t \equiv 2 \) (mod 4) and let \( \alpha, \beta \in A_t \) such that \( \alpha \neq \text{id} \) and \( \beta \) is either a cycle of odd length \( k \) or is the product of two disjoint even-length cycles of total length \( k \). Then, there is a sequence \( f_1, \ldots, f_\ell \in (\text{Conj} \cup \text{Comm}) \) such that \( f(\alpha) = \beta \), where \( f := f_\ell \circ \cdots \circ f_1 \) and \( \ell = \log k + O(1) \).

Any function given by this lemma yields a 1-local function \( f : (A_t)^m \to (A_t)^{m - 2^\ell} \) computable in depth \( O(\log t) \) that maps \( \alpha \)-products to \( \beta \)-products while preserving the identity. We now give the proof assuming this lemma.

Proof of Theorem A.0.1. We prove the case \( m = 1 \), but the argument extends immediately to any \( m \). Fix \( \alpha, \beta \in A_t \), and consider the unique representation of \( \beta \) as a set of disjoint cycles \( C = \{ \sigma_1, \ldots, \sigma_\ell \} \subset S_t \). (Here \( C \) contains only those cycles with length \( > 1 \).) The idea is to apply Lemma A.0.3 to each cycle \( \sigma \in C \), obtaining \( f' : A_t \to (A_t)^{O(|\sigma|)} \) such that \( \prod_i f'(\alpha)_i = \sigma \). Then letting \( f \) output the concatenation of these \( f' \), the resulting function maps \( \alpha \)-products to \( \beta \)-products while preserving the identity. Further, its output length is \( \sum_{\sigma \in C} O(|\sigma|) = O(t) \).

The only technical complication has to do with cycles of even length: if \( \sigma \) is a cycle of even length then it is an odd permutation, and so there can be no composition of functions from \( \text{Conj} \) and \( \text{Comm} \) that maps \( \alpha \mapsto \sigma \). We handle this by pairing the cycles of even length, which
we can do because $C$ must contain an even number of cycles of even length as each is an odd permutation and $\beta$ is an even permutation. So, for each such pair of even-length cycles $\sigma, \sigma' \in C$, we instead apply Lemma A.0.3 to get a function $f'$ such that $\prod_i f'_i(\alpha)_i = \sigma \cdot \sigma'$.

To prove Lemma A.0.3, we implement the procedure described above. Namely, we first use $\leq 2$ commutations to convert a given $\alpha \in A_t$ to a double-transposition, then $\log k$ commutations to convert to a product of roughly $k/2$ disjoint transpositions, and finally $\leq 2$ commutations and 1 conjugation to convert to either a cycle of odd length $k$ or to the product of two disjoint even-length cycles with total length $k$. The first step is actually Lemma 3.3.20 which was proved in §3.3.5.1.

**Lemma A.0.4.** For every double-transposition $\alpha \in A_t$ and even $k \leq t/2$, there exist $\gamma_1, \ldots, \gamma_{\log k} \in A_t$ such that $[[\ldots[[\alpha, \gamma_1], \gamma_2], \ldots], \gamma_{\log k}]$ is a product of $k$ disjoint transpositions.

**Proof.** Given $\alpha = (a b)(c d)$, we can double the number of transpositions by commutating with $\gamma = (a e)(b f)(c g)(d h)$ to get $[\alpha, \gamma] = (a b)(c d)(e f)(g h)$. Repeating this $\log(k) - 1$ times (with appropriate modifications to $\gamma$) grows the number of transpositions from 2 to $k$. To handle $k$ that is not a power of 2, note that any $(a b)(c d)$ can be “maintained” rather than doubled by instead commutating with $(a b c)$ as in the proof of Lemma 3.3.20.

**Lemma A.0.5.** Let $t$ be even and $\beta \in A_t$ be any cycle of odd length $5 \leq k \leq t - 1$. For any $\alpha \in A_t$ that is the product of either $(k - 1)/2$ or $(k - 3)/2$ disjoint transpositions (depending on which is even), there are $\gamma_1, \gamma_2, \gamma_3 \in A_t$ such that either $[\gamma_1^{-1} \alpha \gamma_1, \gamma_2] = \beta$ or $[[\gamma_1^{-1} \alpha \gamma_1, \gamma_2], \gamma_3] = \beta$.

**Proof.** We first show that $\alpha$ can be converted to a $k$-cycle with $\leq 2$ commutations. Afterwards we observe that by first using 1 conjugation, these commutations can be made to produce the specific $k$-cycle $\beta$.

If $(k - 1)/2$ is even, then let $\alpha := (a_1 b_1) \cdots (a_{k'} b_{k'})$ be any product of $k' := (k - 1)/2$ disjoint transpositions. Choosing $\gamma := (a_1 b_1 a_2 b_2 \cdots a_{k'} b_{k'} c) \in A_t$, where $c$ is distinct from all $k - 1$ points permuted by $\alpha$, we get that

$$[\alpha, \gamma] = (a_1 \cdots a_{k'} b_{k'} \cdots b_1 c)$$

is a $k$-cycle. (We only need one commutation in this case.)

If instead $(k - 3)/2$ is even (so $k \geq 7$), then let $\alpha$ be any product of $(k - 3)/2$ disjoint transpositions. First, commutate once as above to get a $(k - 2)$-cycle

$$\mu := (a_1 \cdots a_{k-2}).$$
We now show that there is another \((k - 2)\)-cycle \(\pi \in A_t\) such that \(\mu \pi\) is a \(k\)-cycle. This implies that we can convert \(\mu\) to a \(k\)-cycle with one more commutation, namely by commutating with \(\gamma \in A_t\) such that \(\gamma \mu^{-1} \gamma^{-1} = \pi\). (Such \(\gamma\) must exist because the set of \((k - 2)\)-cycles forms a conjugacy class in \(A_t\) when \(t \geq k + 1\).) Take the \((k - 2)\)-cycle

\[
\pi := (a_1 \ c_1 \ c_2 \ c_3 \ a_4 \cdots \ a_{k-5} \ a_2)
\]

where \(a_1, c_2, c_3\) are distinct from the \(k - 2 \leq t - 3\) points permuted by \(\mu\). Then letting \(a_i \cdots a_j\) denote the sequence \(a_i \ a_{i+2} \ a_{i+4} \cdots a_j\), we have that

\[
\mu \pi = (a_2 \cdots a_{k-5} a_{k-4} a_{k-3} a_{k-2} \ c_1 \ c_2 \ c_3 \cdots a_{k-6})
\]

is a \(k\)-cycle (permuting points \(a_2, \ldots, a_{k-2}, c_1, c_2, c_3\)).

Having converted \(\alpha\) to a \(k\)-cycle with \(\leq 2\) commutations, one might hope to then use 1 conjugation to convert to the specific \(k\)-cycle \(\beta\). However when \(k = t - 1\), the \(k\)-cycles form two distinct conjugacy classes in \(A_t\) so we cannot do this. We instead note that the points permuted by the \(k\)-cycle depend directly on the points permuted by \(\alpha\) (and the extra points \(c_i\)), and that products of an equal number of disjoint transpositions are conjugate in \(A_t\). So by first using 1 conjugation to modify \(\alpha\) appropriately, the above commutations yield \(\beta\).

**Lemma A.0.6.** Let \(t \equiv 2 \pmod{4}\) and \(\beta \in A_t\) be any product of two disjoint cycles of even lengths \(k_1, k_2\). Denote \(k = k_1 + k_2\). For any \(\alpha \in A_t\) that is the product of either \(k/2\) or \(k/2 - 1\) disjoint transpositions (depending on which is even), there exist \(\gamma_1, \gamma_2 \in A_t\) such that \(\gamma_2^{-1} \cdot [\alpha, \gamma_1] \cdot \gamma_2 = \beta\).

**Proof.** We first use one commutation to convert \(\alpha\) to the product of a \(k_1\)-cycle and a \(k_2\)-cycle, and then one conjugation to convert it to \(\beta\) (which here we can do without the complication mentioned at the end of Lemma A.0.5). We assume that \(k_1, k_2 \geq 4\), and at the end mention how to handle the two cases \(k_1 = k_2 = 2\) and \(k_1 = 2, k_2 = 4\).

If \(k/2\) is even, let

\[
\alpha := (a_1 \ b_1) \cdots (a_{k_1'} \ b_{k_1'}) (c_1 \ d_1) \cdots (c_{k_2'} \ d_{k_2'})
\]

be any product of \(k/2 = k_1' + k_2'\) disjoint transpositions, where \(k_1' := k_1/2\) and \(k_2' := k_2/2\). We will show that there exists \(\pi \in A_t\) that is the product of \(k/2\) disjoint transpositions such that \(\alpha \pi\) is the product of a \(k_1\)-cycle and a \(k_2\)-cycle. As in Lemma A.0.5, this implies that we can convert \(\alpha\) to the desired form by commutating with \(\gamma \in A_t\) such that \(\gamma \alpha^{-1} \gamma^{-1} = \pi\). Define

\[
\pi := \prod_{i=1}^{k_1'-1} (a_{i+1} \ b_i) \cdot \prod_{i=1}^{k_2'-2} (c_{i+1} \ d_i) \cdot (d_{k_2'-1} \ e_1) (c_{k_2'} \ d_{k_2'}) (c_1 \ e_2)
\]
where \( e_1, e_2 \in [t] \) are distinct from each point permuted by \( \alpha \). (Such \( e_1, e_2 \) must exist because \( 4k \) and \( t \equiv 2 \pmod{4} \), and thus \( k \leq t - 2 \).) Then we have that 

\[
\alpha \pi = (a_1 \cdots a_{k'_1} b_{k'_1} \cdots b_1)(c_1 \cdots c_{k'_2-1} e_1 d_{k'_2-1} \cdots d_1 e_2)
\]

is the product of two disjoint cycles of lengths \( 2k'_1 = k_1 \) and \( 2k'_2 = k_2 \).

If instead \( k/2 - 1 \) is even, let 

\[
\alpha := (a_1 b_1) \cdots (a_{k'_1} b_{k'_1})(c_1 d_1) \cdots (c_{k'_2-1} d_{k'_2-1})
\]

be any product of \( k/2 - 1 = k'_1 + k'_2 - 1 \) disjoint transpositions. This time we define 

\[
\pi := \prod_{i=1}^{k'_1-1} (a_i+1 b_i) \cdot \prod_{i=1}^{k'_2-3} (c_i+1 d_i) \cdot (c_1 c_{k'_2-1})(d_{k'_2-2} e_1)(d_{k'_2-1} e_2)
\]

where again \( e_1, e_2 \) are distinct from each of the \( 2(k/2 - 1) \leq t - 2 \) points permuted by \( \alpha \) (here we use \( k'_1 \geq 2, k'_2 \geq 3 \)). Then we have that 

\[
\alpha \pi = (a_1 \cdots a_{k'_1} b_{k'_1} \cdots b_1)(c_1 \cdots c_{k'_2-2} e_1 d_{k'_2-2} \cdots d_1 c_{k'_2-1} e_2 d_{k'_2-1})
\]

is the product of two disjoint cycles of lengths \( 2k'_1 = k_1 \) and \( 2k'_2 = k_2 \).

Finally we handle the two cases \( k_1 = k_2 = 2 \) and \( k_1 = 2, k_2 = 4 \). If \( k_1 = k_2 = 2 \) then \( \alpha \) and \( \beta \) are both double-transpositions and can be made equal with a single conjugation. Otherwise denote \( \alpha = (a b)(c d) \), and note that there is another double-transposition \( \pi = (c e)(d f) \) such that \( \alpha \pi = (a b)(c f d e) \). Thus \( \alpha \) can be commutated to the product of a 2-cycle and a 4-cycle, and a conjugation can make it equal to \( \beta \). \( \square \)

**Proof of Lemma A.0.3.** The proof follows immediately from Lemma 3.3.20 and the three lemmas proved above. The only cases not explicitly covered by these are when \( t = 2 \) in which case Lemma A.0.3 is vacuous, and when \( \beta \) is a 3-cycle (because Lemma A.0.5 only handles \( (k \geq 5) \)-cycles). For the latter, note that by assumption we must have \( t \geq 6 \). We first convert \( \alpha \) to a 5-cycle \( (a_1 \cdots a_5) \) using the above lemmas, then use one commutation with \( (a_2 a_3 a_4) \) to convert to \( (a_1 a_4 a_3) \), and finally use one conjugation to convert to \( \beta \). \( \square \)