On the Invalidity of Lin16/Lin17 Obfuscation Schemes

Yupu Hu[1✉], Siyue Dong[1], Baocang Wang[1], and Xingting Dong[2]

[1] State Key Laboratory of Integrated Services Networks,
Xidian University, Xi'an, Shaanxi, China
yphu@mail.xidian.edu.cn
[2] School of Computer Science and Information Security, Guilin University Of Electronic Technology,
Guilin, Guangxi, China

Abstract. Indistinguishability obfuscation (IO) is at the frontier of cryptography research. Lin16/Lin17 obfuscation schemes are famous progresses towards simplifying obfuscation mechanism. Their basic structure can be described in the following way: to obfuscate a polynomial-time-computable Boolean function $c(x)$, first divide it into a group of component functions with low-degree and low-locality by using randomized encoding, and then hide the shapes of these component functions by using constant-degree multilinear maps (rather than polynomial degree ones).

In this short paper we point out that Lin16/Lin17 schemes are invalid. More detailedly, they cannot achieve reusability, therefore they are not true IO schemes, but rather garbling schemes which are one-time schemes. Besides, this short paper presents more observations, to show that component functions cannot be overly simple.

Keywords: Indistinguishability obfuscation · Multilinear maps · Yao's garbling · Randomized encoding.

1 Introduction

Both indistinguishability obfuscation (IO) and garbling can make the function unintelligent which, for chosen value of independent variable, provides nothing except the function value. In the basic scene of these two cryptographic primitives there are two sides, encoding-side and decoding-side. Encoding-side presents unintelligent form $\overline{c}$ of the function c, while decoding-side chooses value x and computes $\overline{c}(x)$ (which equals to $c(x)$). The unique difference is that an IO scheme is a "reusable scheme" while a garbling scheme is a "one-time scheme". In IO schemes, decoding-side can repeatedly use $\overline{c}$ to compute $\overline{c}(x)$ for different x of his choice, without contacting encoding-side for each computation. In garbling schemes $\overline{c}$ can be used only once, otherwise the security is not guaranteed. (There was a "reusable garbling scheme" [1,2] but we pointed [3] that it is still a one-time scheme). This is the reason of following three facts.

Fact 1 Garbling is much easier to be constructed than IO. The former does not use novel cryptographic tools, while the latter needs multi-linear maps, maybe fully-homomorphic encryption, zero-knowledge proof, and so on.

Fact 2 Garbling is much more mature than IO. The former has complete security proof, while each candidate of the latter is not clear about the security.

Fact 3 Garbling has much less applications than IO. The former is mainly applied for multi-party computation and some similar scenes, while the latter, if exists, is a revolutionary advance in public-key cryptography.

IO was first defined by Barak [4], and has received a lot of attentions in the community [5–29]. Early candidates of IO schemes directly make use of multilinear maps [30], which is a novel technology. However it is extremely huge in size. Since then, many efforts have been made on simplifying obfuscation mechanism, among which Lin16 and Lin17 obfuscation schemes [16, 18] are famous progresses. Their basic structure can be described in the following way: in order to obfuscate any polynomial-time-computable boolean function $c(x)$, they first divide this $c(x)$ into a group of component functions with low-degree and low-locality functions, and then hide the shapes of all these component functions through using constant-degree multilinear maps. The most important novelty of Lin16/Lin17 schemes is the significant bootstrapping process from constant-degree multilinear maps, rather than polynomial-degree ones. Besides, one essential ingredient in the Lin16/Lin17 construction is the usage of randomized encoding, which can be instantiated by IK00/IK02/AIK04/AIK06 [31–34]. Besides, Lin17 scheme has an additional step to increase the number of variables so as to decrease the degree of component function, therefore Lin17 scheme uses lower degree multi-linear maps than Lin16 scheme.

In this short paper we point out that Lin16/Lin17 schemes are invalid. More detailedly, they cannot achieve reusability, therefore they are not true IO schemes, but rather garbling schemes which are one-time schemes. The reason is that randomized encoding, a ground structure of Lin16/Lin17 schemes, is not reusable, and that the computation procedure of the decoding-side can obtain {input, output} of such randomized encoding.

Besides, this short paper presents more observations to a large class of designing ideas including Lin16/Lin17 schemes, to show that component functions cannot be overly simple.

2 Preliminaries: IO, Graded Encoding, Garbling, and Randomized Encoding

2.1 Definition and Two Notes of Indistinguishability Obfuscation (IO)

Definition 2.1 *A uniform PPT machine IO is called an indistinguishability obfuscator [5] for a circuit class $\{C_\lambda\}$ if the following two conditions are satisfied:*

(1) Correctness. *For all security parameters λ, for all circuits $c \in C_\lambda$, for all inputs x,*$\Pr\left[\overline{c}\left(x\right) = c\left(x\right) : \overline{c} \leftarrow IO\left(\lambda, c\right)\right] = 1$

(2) Indistinguishability. *For any PPT distinguisher D, there exists a negligible function α such that the following holds. For all security parameters λ, for all pairs $c_0, c_1 \in C_\lambda$, we have that if $c_0(x) = c_1(x)$ for all inputs x, then* $|\Pr[D(IO(\lambda, c_0)) = 1] - \Pr[D(IO(\lambda, c_1)) = 1]| \leq \alpha(\lambda)$.

Note 1. $\overline{c}$ should be reusable. That is, once $\overline{c}$ is constructed, it should be fixed and repeatedly used for computing $\overline{c}(x)\,(= c(x))$ of different values of x. If $\overline{c}$ is used only once, it is much easier to be constructed, and is the component of garbling, a weaker primitive.

Note 2. $\overline{c}$ should be a "real white box" and an "essential black box". First, most applicable $\overline{c}$ is white box, that is, each component of $\overline{c}$ is clearly defined, without any black box implementation. For example, a previously published truth table is a white box obfuscator. If $\overline{c}$ is not a white box obfuscator, it is hoped to be as white as possible. The whiter $\overline{c}$ would be, the more applicable it is. Second, $\overline{c}$ should be essential black box, that is, $\overline{c}$ leaks nothing about c except $\overline{c}(x) = c(x)$.

2.2 Graded Encoding

Graded encoding (also called multi-linear map) [40, 41] is the most important component of IO and sometimes unique component (That is, sometimes graded encoding itself is IO).

For a Boolean circuit c, operations of encoding-side are as follows. He encodes c into $\overline{c}$, and encodes independent variable range X into $\overline{X}$ (That is, encodes each value from $\{0,1\}$ of each entry X_i of $X = (X_1, \cdots, X_n)$. So that $\overline{X} = (\overline{X_{1,0}}, \overline{X_{1,1}}, \cdots, \overline{X_{n,0}}, \overline{X_{n,1}})$). He also constructs the decoding tool T (also called zero-testing tool, which can test value just at one point, neither sooner nor later). Then he submits $\{\overline{X}, \overline{c}, T\}$.

Decoding-side obtains $\{\overline{X}, \overline{c}, T\}$, and chooses $\overline{x}$ from $\overline{X}$ according to his choice of x. Then he can compute $T(\overline{x}, \overline{c})$ which is equal to $c(x)$.

The first challenge is to guarantee $T(\overline{x}, \overline{c}) = c(x)$ for any x with leaking nothing else, and the second challenge is to reduce the huge size. All candidates of graded encoding are not very sure to face such two challenges, so that an effort is to ease the role of graded encoding in the IO structure.

2.3 About Garbling and Randomized Encoding

Garbling [31–34, 37, 38] can be taken as a one-time version of IO, so that it is much simpler than IO. Although some garbling schemes [31] have limited ability for small number of reusability, no garbling candidate can be taken as a true reusable scheme. Besides, there was a "reusable garbling scheme" [1, 2], but we pointed [3] it is still a one-time scheme.

A technical difference of garbling is that, $\overline{X}$ is not completely sent to decoding-side, but rather by using "one-out-two oblivious transfer". That is, for each entry X_i of $X = (X_1, \cdots, X_n)$, decoding-side can and only can choose to receive one from $\{\overline{X_{i,0}}, \overline{X_{i,1}}\}$, while encoding-side doesn't know the choice of decoding-side.

Randomized encoding [31–34] is a special type of garbling, to express a function in terms of a group of low-degree low-locality functions with random parameters.

3 Lin16/Lin17 IO Schemes [16] [18]

Suppose c is a polynomial-time-computable Boolean function. The schemes have two stages, the first stage is for operations of encoding-side, and the second for decoding-side.

3.1 Lin16 Scheme: Operations of Encoding-side

Step 1 (garbling)Use Yao's garbling of c [1,31,32,37–40] to construct I Boolean functions $c_i^* (x,k) = Yao_i (x, PRF(k))$, $i \in \{1,2,\cdots,I\}$, where k is randomly chosen, PRF is a pseudorandom function.

Step 2 (randomized encoding)For each $i \in \{1,2,\cdots,I\}$, use AIK randomized encoding of c_i^* [31–34] to construct J Boolean functions $c_{ij}^{**}(x,k,s) = AIK_{ij}(x,k,PRG(s))$, $j \in \{1,2,\cdots,J\}$, where s is randomly chosen, PRG is a low-degree low-locality pseudorandom generator. Notice that AIK_{ij} is a low-degree low-locality Boolean function, therefore $c_{ij}^{**}(x,k,s)$ is a low-degree low-locality Boolean function.

Step 3 For each $i \in \{1,2,\cdots,I\}$, $j \in \{1,2,\cdots,J\}$ define function c_{ij}^{***} as

$$c_{ij}^{***}(x,k,s,b) = \begin{cases} c_{ij}^{**}(x,k,s), & for\ b=0 \\ any\ function, & for\ b=1 \end{cases}$$

The purpose of constructing such c_{ij}^{***} is to make so called "decryption key" complicated enough, so as to hide the shape of c_{ij}^{**}.

Step 4 (graded encoding) Up to now, each c_{ij}^{***} is a low-degree low-locality Boolean function. Then use graded encoding to encode independent variable range X into $\overline{X} = \left(\overline{X_{1,0}}, \overline{X_{1,1}}, \cdots, \overline{X_{n,0}}, \overline{X_{n,1}}\right)$, encode parameters (k,s,b) into $(\overline{k},\overline{s},\overline{b})$, and encode c_{ij}^{***} into $\overline{c_{ij}^{***}}$. Construct decoding tool (zero-testing tool) T, to guarantee that for any x, corresponding $\overline{x} \in \overline{X}$, $T(\overline{x},\overline{k},\overline{s},\overline{b},\overline{c_{ij}^{***}}) = c_{ij}^{***}(x,k,s,b)$. Submit $\{\overline{X},\overline{k},\overline{s},\overline{b},\overline{c_{ij}^{***}},T\}$.

3.2 Lin16 Scheme: Operations of Decoding-side

Step 1 (graded decoding) By obtained $\{\overline{X},\overline{k},\overline{s},\overline{b},\overline{c_{ij}^{***}},T\}$ and chosen x, pick corresponding $\overline{x}$ from $\overline{X}$ and compute $T(\overline{x},\overline{k},\overline{s},\overline{b},\overline{c_{ij}^{***}})(= c_{ij}^{***}(x,k,s,b) = c_{ij}^{**}(x,k,s))$.

Step 2 (randomized decoding) Use $\{c_{ij}^{**}(x,k,s), i=1,\cdots,I, j=1,\cdots,J\}$ to compute $\{c_i^*(x,k), i=1,\cdots,I\}$.

Step 3 (degarbling)Use $\{c_i^*(x,k), i=1,\cdots,I\}$ to compute $c(x)$.

3.3 Lin17 Scheme: Operations of Encoding-side

Step 1∼3 Same as Step 1∼3 of operations of encoding-side of Lin16 scheme (see subsection 3.1).

Step 4 (increasing the number of variables to decrease the degree) Take $x \times x = \{x_u x_v\}$, and we know $x_u \cdot x_u = x_u$. Similarly, take $x \times k = \{x_u k_v\}$, $x \times s = \{x_u s_v\}$, $x \times b = \{x_u b\}$, $k \times k = \{k_u k_v\}$, $k \times s = \{k_u s_v\}$, $k \times b = \{k_u b\}$, $s \times s = \{s_u s_v\}$, $s \times b = \{s_u b\}$. For each $i \in \{1, \cdots, I\}$, $j \in \{1, \cdots, J\}$, express c_{ij}^{***} as the function of $(x \times x, x \times k, x \times s, x \times b, k \times k, k \times s, k \times b, s \times s, s \times b, b)$, rather than only the function of (x, k, s, b). That is, take an expression $c_{ij}^{****}(x \times x, x \times k, x \times s, x \times b, k \times k, k \times s, k \times b, s \times s, s \times b, b) = s_{ij}^{***}(x, k, s, b)$, then c_{ij}^{****} has a lower degree than c_{ij}^{***}.

Step 5 (graded encoding) Each entry of $x \times x$ should have graded encodings for 0 and 1 respectively, therefore graded encoding of $x \times x$ is denoted by $\overline{X \times X} = (\overline{X \times X}_{(1,1),0}, \overline{X \times X}_{(1,1),1}, \overline{X \times X}_{(1,2),0}, \overline{X \times X}_{(1,2),1}, \cdots, \overline{X \times X}_{(n,n),0}, \overline{X \times X}_{(n,n),1})$. Each entry of $x \times k$ should have graded encodings for $0k_v$ and $1k_v$ respectively, therefore graded encoding of $x \times k$ is denoted by $\overline{X \times k} = (\overline{X \times k}_{(1,1),0}, \overline{X \times k}_{(1,1),1}, \overline{X \times k}_{(1,2),0}, \overline{X \times k}_{(1,2),1}, \cdots, \overline{X \times k}_{(n,I),0}, \overline{X \times k}_{(n,I),1})$. In other words, although $0k_v = 1k_v = 0$ if $k_v = 0$, we should still take two graded encodings for $0k_v$ and $1k_v$ respectively, because k_v should not be known by decoding-side. Similar cases are $\overline{X \times s}$ and $\overline{X \times b}$. Each entry of $k \times k$ should have only one graded encoding for just one value $k_u k_v$, therefore graded encoding of $k \times k$ is denoted by $\overline{k \times k} = (\overline{k \times k}_{(1,1)}, \overline{k \times k}_{(1,2)}, \cdots, \overline{k \times k}_{(I,I)})$. Similar cases are $\overline{k \times s}$, $\overline{k \times b}$, $\overline{s \times s}$, $\overline{s \times b}$, $\overline{b}$. Graded encoding of c_{ij}^{****} is denoted by $\overline{c_{ij}^{****}}$. Construct decoding tool (zero-testing tool) T. Submit $\{\overline{X \times X}, \overline{X \times k}, \overline{X \times s}, \overline{X \times b}, \overline{k \times k}, \overline{k \times s}, \overline{k \times b}, \overline{s \times s}, \overline{s \times b}, \overline{b}, \{\overline{c_{ij}^{****}}\}, T\}$.

3.4 Lin17 Scheme: Operations of Decoding-side

Step 1 (graded decoding) By received $\{\overline{X \times X}, \overline{X \times k}, \overline{X \times s}, \overline{X \times b}, \overline{k \times k}, \overline{k \times s}, \overline{k \times b}, \overline{s \times s}, \overline{s \times b}, \overline{b}, \{\overline{c_{ij}^{****}}\}, T\}$ and chosen x, pick corresponding $\overline{x \times x}$ from $\overline{X \times X}$, $\overline{x \times k}$ from $\overline{X \times k}$, $\overline{x \times s}$ from $\overline{X \times s}$, and $\overline{x \times b}$ from $\overline{X \times b}$. Then compute $T(\overline{x \times x}, \overline{x \times k}, \overline{x \times s}, \overline{x \times b}, \overline{k \times k}, \overline{k \times s}, \overline{k \times b}, \overline{s \times s}, \overline{s \times b}, \overline{b}, \overline{c_{ij}^{****}}) = c_{ij}^{****}(x \times x, x \times k, x \times s, x \times b, k \times k, k \times s, k \times b, s \times s, s \times b, b) = c_{ij}^{***}(x, k, s, b) = c_{ij}^{**}(x, k, s)$.

Step 2∼3 Same as Step 2∼3 of operations of decoding-side of Lin16 scheme (see subsection 3.2).

4 The Invalidity of Lin16/Lin17 Schemes

From operations of decoding-side, he can obtain values of $c_{ij}^{**}(x, k, s)$ and $c_i^*(x, k)$ for his choice of x. If the secret parameters (k, s) are reused for different x, the garbling circuit $\{c_i^*, i = 1, \cdots, I\}$ are reused, which is not secure. If (k, s) are used only once, Lin16/Lin17 schemes are nothing different with a garbling scheme.

It may be considered that (k, s) are kept fixed, and $\{c_{ij}^{**}(x, k, s), c_i^*(x, k)\}$ unknown by decoding-side. It seems that the unique method is to integrate operations of decoding-side into a black box, so that he can only input x and obtain $c(x)$. However, such "IO" is far from the original idea, and much less applicable.

5 More Observations

Besides Lin16/Lin17 schemes, several works for simplifying IO [5,19,23,27] have similar idea which, for a polynomial-time-computable function, first to express it into a group of simple component functions, second to hide shapes of these component functions (by using low-degree multi-linear maps or some other cryptographic tools). Now always suppose that component functions are reusable, and that decoding-side can repeatedly choose values of independent variable.

Our first observation When component functions are overly simple, their shapes cannot be hidden, and can be computed by several chosen values of independent variable. An example is low-degree low-locality functions in Lin16/Lin17 schemes.

Our second observation Up to now, there is no such IO scheme, for that shapes of component functions cannot be hidden while that of original function can. For Lin16/Lin17 schemes, leakage of shapes of component functions implies leakage of that of original functions, because the protection of garbling and randomized encoding is limited to one-time using.

Our third observation IO scheme of Garg et al [5] is a successful example. The shapes of component functions are not leaked. Why? Because that component functions are from NC^1 ("verifying circuit of fully-homomorphic evaluation" + "fully-homomorphic decryption"), and that relation between component functions and original function is covered by fully-homomorphic encryption (FHE). The advantage of expressing a polynomial-time-computable function into a group of NC^1 functions is still decreasing the degree of multi-linear maps, but the decreased degree is not "constant degree".

References

1. Goldwasser, S., Kalai, Y.T., Popa, R.A., Vaikuntanathan, V., Zeldovich, N.: Reusable garbled circuits and succinct functional encryption. In: STOC, pp. 555–564, 2013.
2. Agrawal, S.: Stronger security for reusable garbled circuits, general definitions and attacks. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10401, pp. 3-35. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63688-7_1
3. Hu, Y, P., Dong, S, Y., Wang, B, C., Liu, J.: Notes on Reusable Garbling. Cryptology ePrint Archive, Paper 2022/1208.https://eprint.iacr.org/2022/1208
4. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., Yang, K.: On the (im)possibility of obfuscating programs. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 1-18. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44647-8_1

5. Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., Waters, B.: Candidate indistinguishability obfuscation and functional encryption for all circuits. In: 54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October 2013, Berkeley, CA, USA, pp. 40-49 (2013).
6. Brakerski, Z., Rothblum, G.N.: Virtual black-box obfuscation for all circuits via generic graded encoding. In: Lindell, Y. (ed.) TCC 2014. LNCS, vol. 8349, pp. 1-25. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54242-8_15
7. Barak, B., Garg, S., Kalai, Y.T., Paneth, O., Sahai, A.: Protecting obfuscation against algebraic attacks. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 221-238. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_13
8. Pass, R., Seth, K., Telang, S.: Indistinguishability obfuscation from semantically secure multilinear encodings. In: Gary, J.A., Gennaro, R. (eds.) CRYPTO 2014. LNCS, vol. 8616, pp. 500-517. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44371-2_28
9. Ananth, P.V., Gupta, D., Ishai, Y., Sahai, A.: Optimizing obfuscation: avoiding Barrington's theorem. In: ACM CCS 2014, Scottsdale, AZ, USA, pp. 646-658, 3-7 November 2014.
10. Applebaum, B., Brakerski, Z.: Obfuscating circuits via composite-order graded encoding. In: Y. Dodis, Nielsen, J.B. (eds.) TCC 2015. LNCS, vol. 9015, pp. 528-556. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46497-7_21
11. Zimmerman, J.: How to obfuscate programs directly. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 439-467. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_15
12. Ananth, P., Jain, A.: Indistinguishability obfuscation from compact functional encryption. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015, Part I. LNCS, vol 9215, pp 308-326. Springer, Berlin, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47989-6_15
13. Gentry, C., Lewko, A.B., Sahai, A., Waters, B.: Indistinguishability obfuscation from the multilinear subgroup elimination assumption. In: Guruswami, V. (ed.) 56th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2015, 17-20 October 2015, Berkeley, CA, USA, pp. 151-170 (2015).
14. Bitansky, N., Vaikuntanathan, V.: Indistinguishability obfuscation from functional encryption. In: Guruswami, V. (ed.) 56th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2015, 17-20 October 2015, Berkeley, CA, USA, pp. 171–190(2015).
15. Miles, E., Sahai, A., Zhangdry, M.: Annihilation attacks for multilinear maps: cryptanalysis of indistinguishability obfuscation over GGH13. In: IACR Cryptology ePrint Archive, vol. 2016, p. 147 (2016).
16. Lin, H.: Indistinguishability obfuscation from constant-degree graded encoding schemes. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, LNCS, vol.9665, pp. 28–57. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49890-3_2
17. Ananth, P., Sahai, A.: Projective arithmetic functional encryption and indistinguishability obfuscation from degree-5 multilinear maps. In: Coron, JS., Nielsen, J. (eds.) EUROCRYPT 2017, Part I. LNCS, vol 10210, pp 152-181. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56620-7_6
18. Lin, H.: Indistinguishability obfuscation from SXDH on 5-linear maps and locality-5 PRGs. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part I. LNCS, vol.10401, pp. 599-629. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-319-63688-7_20

19. Lin, H., Tessaro, S.: Indistinguishability obfuscation from trilinear maps and blockwise local PRGs. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part I. LNCS, vol. 10401, pp. 630–660. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63688-7_21
20. Ananth, P., Jain, A., Sahai, A.: Indistinguishability obfuscation without multilinear maps: iO from LWE, bilinear maps, and weak pseudorandomness. Cryptology ePrint Archive, Report 2018/615 (2018).
21. Gentry, C., Jutla, C.S., Kane, D.: Obfuscation Using Tensor Products. In: Electronic Colloquium on Computational Complexity (ECCC), vol. 25 (2018).
22. Lin, H., Matt, C.: Pseudo Flawed-Smudging Generators and Their Application to Indistinguishability Obfuscation. Cryptology ePrint Archive, Report 2018/646 (2018).
23. Agrawal, S.: Indistinguishability obfuscation without multilinear maps: new methods for bootstrapping and instantiation. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part I. LNCS, vol 11476, pp. 191-225. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17653-2_7
24. Jain, A., Lin, H., Matt, C., Sahai, A.: How to leverage hardness of constantdegree expanding polynomials over R to build iO. In: Ishai, Y., Rijmen, V.(eds.) EUROCRYPT 2019, Part I. LNCS, vol. 11476, pp. 251-281. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17653-2_9
25. Bartusek, J., Lepoint, T., Ma, F., Zhandry, M.: New techniques for obfuscating conjunctions. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part III. LNCS, vol 11478, pp 636-666. Springer, Cham(2019). https://doi.org/10.1007/978-3-030-17659-4_22
26. Ananth, P., Jain, A., Lin, H., Matt, C., Sahai, A.: Indistinguishability obfuscation without multilinear maps: new paradigms via low degree weak pseudorandomness and security amplification. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part III. LNCS, vol 11694, pp 284-332. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26954-8_10
27. Agrawal, S., Pellet-Mary, A.: Indistinguishability obfuscation without maps: attacks and fixes for noisy linear FE. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part I. LNCS, vol 12105, pp. 110-140. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45721-1_5
28. Brakerski, Z., Döttling, N., Garg, S., Malavolta, G.: Candidate iO from homomorphic encryption schemes. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part I. LNCS, vol 12105, pp. 79-109. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45721-1_4
29. Bartusek, J., Ishai, Y., Jain, A., Ma, F., Sahai, A., Zhandry, M.: Affine determinant programs: A framework for obfuscation and witness encryption. In: 11th Innovations in Theoretical Computer Science Conference (ITCS 2020). Schloss Dagstuhl-Leibniz-Zentrum für Informatik, (2020).
30. Garg, S., Gentry, C., Halevi, S.: Candidate multilinear maps from ideal lattices. In: T. Johansson, P.Q. Nguyen (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 1-17. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_1
31. Ishai, Y., Kushilevitz, E.: Randomizing Polynomials: A new representation with applications to round-efficient secure computation. In: Proceedings of the 41st FOCS, pp. 294-304 (2000).
32. Ishai, Y., Kushilevitz, E.: Perfect constant-round secure computation via perfect randomizing polynomials. In: Widmayer, P., Eidenbenz, S., Triguero, F., Morales, R., Conejo, R., Hennessy, M., (eds.) ICALP 2002. LNCS, vol. 2380, pp. 244-256. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45465-9_22

33. Applebaum, B., Ishai, Y., Kushilevitz, E.: Cryptography in nc0. In: Proceedings of the 45th FOCS, pp. 166-175 (2004).
34. Applebaum, B., Ishai, Y., Kushilevitz, E.: Computationally private randomizing polynomials and their applications. In: Computational Complexity, vol. 15 (2006), pp. 115 – 162. https://doi.org/10.1007/s00037-006-0211-8
35. Barrington, D.A.: Bounded-width polynomial-size branching programs recognize exactly those languages in NC1. In: Journal of computer and system sciences, vol. 38, no. 1, pp. 150 – 164 (1989), Elsevier 1989.
36. Gorbunov, S., Vaikuntanathan, V., Wee, H.: Attribute-based encryption for circuits. In: Boneh, D., Roughgarden, T., Feigenbaum, J. (eds.) 45th ACM STOC, pp. 545-554. ACM Press, June 2013.
37. Yao, A.C.: Protocols for secure computations (extended abstract). In: Proceedings of the 23th FOCS, pp. 160-164 (1982).
38. Yao, A.C.: How to generate and exchange secrets (extended abstract). In: Proceedings of the 27th FOCS, pp. 162-167 (1986).
39. Bellare, M., Hoang, V.T., Rogaway, P.: Foundations of garbled circuits. In: Proceedings of the 2012 ACM conference on Computer and Communications Security (CCS), pp. 784-796. October 2012.
40. Gentry, C., Gorbunov, S., Halevi, S., Vaikuntanathan, V., Vinayagamurthy, D.: How to compress (reusable) garbled circuits. In: IACR eprint 2013/687.
41. Coron, J.-S., Lepoint, T., Tibouchi, M.: Practical multilinear maps over the integers. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 476–493. Springer, Heidelberg (2013)