Secure IP-Block Distribution for Hardware Devices

Jorge Guajardo*, Tim Güneysu†, Sandeep S. Kumar*, Christof Paar†
*Philips Research Europe, Eindhoven, The Netherlands
Email: {jorge.guajardo,sandeep.kumar}@philips.com
†Horst Görtz Institute for IT Security, Ruhr University Bochum, Germany
Email: {christof.paar,tim.gueneysu}@crypto.rub.de

***Abstract*—EDA vendors have proposed a standard for the sharing of IP among vendors to be used in the design and development of IP for FPGAs. Although, we do not propose any attacks, we show that there are easy ways in which the security of the whole process can be enhanced by using standard cryptographic techniques such as secret sharing and public-key based key exchange. We also explore the advantages that newer primitives have such as All-Or-Nothing Transforms and Physical Unclonable Functions. We show that the protocols proposed would significantly reduce the effects that the leakage of a single key would have over the whole system.**

Index Terms — **Security, IP Protection, EDA Tools, FPGAs, Secret Sharing Schemes, All-Or-Nothing Transforms**

I. Introduction

Field Programmable Gate Arrays (FPGAs) have gained widespread acceptance as substitutes for ASICs in many applications. In fact their re-programmability has made them very attractive in the embedded market, where software and functionality updates are common and desirable by customers. As a result of this shift, it is increasingly the case that the functionality of an embedded system is presented in the form of a bit configuration file or, in the case of microprocessors, in the form of a program. Notice that the very property that makes FPGAs so attractive (their programmability) also makes it very easy for counterfeiters to copy an IP developer's configuration file and create a similar product without the up-front cost of Intellectual Property (IP) development. This problem has been recognized most recently by Simpson and Schaumont [1], however, early references to the problem and suggested solutions date back to at least the late 90's (e.g. [2], [3]). Simpson and Schaumont [1] showed that by using a Physical Unclonable Function (PUF) on an FPGA they could develop protocols that allow binding of a particular IP to a particular FPGA. Their protocols also allow proving authenticity of the IP to the hardware platform. Guajardo et al. [4] reduced the computation and communication complexity of the protocols in [1] and introduced the idea of Intrinsic-PUFs based on the start-up values of SRAM memory values. Both works based their protocols on symmetric-key primitives. In [5], the authors observe that by introducing public-key cryptography, the corresponding private-key does not need to ever leave the FPGA, even during the enrollment stage, thus increasing the security of the overall system. A common characteristic of all PUF-based protocols in [1], [4], [5] is the derivation of a key(s) from the PUF, which is used to encrypt a piece of IP and authenticate its origin. In the remainder of the paper, we will refer to the encrypting operation for ease of presentation but it is clear that our discussion extends to the computation of Message Authentication Codes (MACs) and/or signatures on a particular IP block.

Most of the solutions described until now implicitly assume that the entire IP block programmed in the FPGA has been developed or owned by a single party. However, the fast time-to-market and smaller product cycles have made external silicon intellectual property (IP) providers for various organizations a very attractive option compared to internal development. Companies which specialize in creating IP for external parties require that their IP is not misused (their internal details remain confidential) since they resell the IP to multiple customers. Such IP vendors also require contractual obligations for pricing based on the usage model. Notice that as a consequence of this "IP market," in a real FPGA development environment, the system integrators (who program the FPGA) do not own the entire IP. Thus, it is clear that there is a need for security solutions aimed at protecting the interests of the IP providers.

Electronic Design Automation (EDA) tool providers have recognized this need. As a result, Synplicity suggested an open IP encryption standard [6] aimed at guaranteeing that the IP blocks developed by third parties are protected throughout the design cycle. Notice that the standard allows for the integration of multiple IP blocks originating from different IP providers into a single design. More recently, Drimer et al. [7], propose a solution to the integration of multiple IP blocks in a single design by modifying (albeit the modifications are minor) the FPGA fabric to allow for multiple keys (as opposed to a single key as is common in today's FPGAs) to be securely stored. The work of Drimer et al. builds on work by Güneysu et al. [8], who introduce the concept for single blocks. Their approach makes use of keys derived from secrets and identifying information for a particular FPGA. They make use of public-key cryptography for this purpose. Their methods seem appropriate for FPGAs with secure non-volatile storage and enough resources to implement public-key crypto primitives.

A. Our Contributions

In this paper, we focus on strengthening the security of Synplicity's open IP flow [6]. In particular, we notice that the leakage of a single key from a single EDA tool in the whole chain, would have disastrous consequences. We thus

try to minimize the effects that such leakage would have for the whole system. We emphasize that we do *not* propose an attack against the standard in [6]. From a security perspective, minimizing risks makes sense, specially if the system will be used extensively and throughout the industry, as such a standard would. We show that we can improve the security of the system significantly using standard cryptographic schemes as well as newer primitives which are gaining acceptance such as Physical Unclonable Functions.

B. Organization

The remainder of this contribution is organized as follows. Section II provides the reader an overview of notation, background information on the Diffie-Hellman key exchange protocol and Physical Unclonable Functions (PUFs). The aim is to be self contained for the reader not familiar with these concepts. Then in Sect. III, we describe in detail the core distribution problem, the protocols proposed in [6] and some potential weaknesses of the scheme. We end this section specifying some requirements that we would like the new flow to comply with, for purposes of enhanced security. Sections IV and V describe solutions based on symmetric-key and public-key cryptography, respectively. Section V introduces a protocol, which makes use of PUFs to link the IP block to the hardware platform. We contend that this is the most promising way of guaranteeing pay-per use licensing models for FPGAs. We end with some conclusions in Sect. VII.

II. Preliminaries

In the following, we will assume familiarity with standard cryptographic blocks such as symmetric-key primitives (e.g. the AES, DES, triple-DES), hash functions (e.g. MD5, SHA-1, SHA-2) and public-key based primitives (RSA, elliptic curves). We will denote the operation of encrypting a value X using a symmetric cipher scheme (e.g. AES in CBC mode) and key K by $\mathcal{SE}(K;X)$ and similarly for asymmetric ciphers as $\mathcal{AE}(K_{pub};X)$. The corresponding inverse operations (decryption) will be written as $\mathcal{SE}^{-1}(K;Y)$ and $\mathcal{AE}^{-1}(K_{priv};Y)$, respectively. For authentication in the public-key setting, we will write $\mathcal{S}ig(K_{priv};X)$ for the operation of signing data X with the private key K_{priv} and $\mathcal{V}er(K_{pub};Y)$ for the corresponding verification operation using the public key K_{pub}. When we mean the encryption of a single block using a symmetric-key primitive, we will write $AES(K,X)$, where we have essentially assumed use of the de-facto standard, the AES, as the block cipher of choice. Computing a Message Authentication Code (MAC) or keyed hash function will be written $MAC(K,X)$, while verifying it will be denoted by $MAC^{-1}(K,Y)$. We will write $\{S_i\}_{i=1}^{t}$ to mean the set of S_i's where $i = 1, 2, \cdots, t$.

A. Basic Diffie-Hellman Key Exchange Protocol

Public key cryptography was born with the introduction of Diffie and Hellman's famous key exchange protocol [9]. The key exchange protocol works as follows: Let g be an element of a finite group G of prime order p, then user A generates a random value x, computes $K_A = g^x \bmod p$ and sends K_A to user B, who in turn generates a random value y, computes $K_B = g^y \bmod p$, and sends K_B to A. Both A and B can then compute the session key as $K_{AB} = (K_B)^x = (K_A)^y \equiv g^{xy} \bmod p$. The security of the protocol lies on the hardness of computing a discrete logarithm in a finite group. Notice that we have written the protocol in terms of exponents but the same protocol can be straightforwardly implemented over other structures such as elliptic curves.

B. Physical Unclonable Functions

In 2001, Pappu et al. [10], [11] introduced the concept of Physical Random Functions or Physical Unclonable Functions. Physical Unclonable Functions consist of inherently unclonable physical systems. When a stimulus is applied to the system, it reacts with a response. Such a pair of a stimulus C_i and a response R_i is called a *challenge-response* pair (CRP). Thus, we write: $R_i \leftarrow \mathrm{PUF}(C_i)$. PUFs have essentially two parts: i) a physical part and ii) an operational part. The physical part is a physical system that is very difficult to clone. It inherits its unclonability from uncontrollable process variations during manufacturing. In the case of PUFs on an IC such process variations are typically deep-submicron variations such as doping variations in transistors. Examples of PUFs include optical PUFs [10], silicon PUFs [12], coating PUFs [13] and intrinsic PUFs [4], [14], [15]. The operational part corresponds to a circuit designed to take care of the noise present in PUF responses as well as their non-uniform nature. This is called a helper data algorithm or fuzzy extractor [16], [17]. Efficient implementations of fuzzy extractors have been studied in [18].

C. Parties Involved

Previous works in IP protection for FPGAs [19], [3], [5] have extensively discussed the parties involved in the overall IP protection chain. In this work, we simply make use of previous terminology. In particular, we consider the following parties: the end user, the FPGA customer, the system integrator or designer (SYS), the hardware IP-Provider or core vendor (IPP), the hardware (FPGA) manufacturer (HWM) or vendor, the CAD software or EDA vendor (EDA), and a Trusted Third Party (TTP). Notice that Drimer et al. [7] use similar terminology but different abbreviations for the parties. Also, in the work of [7], the authors assume that the TTP and the HWM correspond to the same party, whereas we allow for them to be different.

III. The Core Distribution Problem

IP providers who build specialized circuits (IP cores) and license them to multiple external parties require that their internal design remains confidential to guarantee profitability. Presently, an IP provider charges a one-time large fee and provides unlimited use of the encrypted IP block to the system integrator. The disadvantage with this approach is that the system integrator has to make a large up front payment to be able to use the IP block. Alternatively, a pay-per-use model

Assumptions and Notation:

- IP provider i has secret key K_{IP_i}
- Each EDA vendor (EDA$_j$) publishes a certified public key $K_{\text{EDA}_{pub_j}}$ and the corresponding private key $K_{\text{EDA}_{priv_i}}$ is embedded in a secure manner in the corresponding tool.
- The connection SYS-IPP is authenticated.

Secure Flow:

1) Transfer of IP block to EDA tools for processing

EDA Tool j — SYS — IPP

$y_i = \mathcal{SE}(K_{IP_i}; IP_i)$

$z_{ji} = \mathcal{AE}(K_{EDApub_j}; K_{IP_i})$

y_i, z_{ij} ← y_i, z_{ij} ←

$K_{IP_i} = \mathcal{AE}^{-1}(K_{EDApriv_j}; z_{ij})$

$IP_i = \mathcal{SE}^{-1}(K_{IP_i}; y_i)$

2) Process IP_i block and generate corresponding flow output N_j. The netlist might be divided into portions N_i corresponding to the original IP_i block.
3) This whole netlist N or some of its components N_i are encrypted again with K_{IP_i} to produce $y_i' = \mathcal{SE}(K_{IP_i}; N_i)$.
4) EDA vendor j tool encrypts key K_{IP_i} with the public-key of EDA vendor k, i.e. $z_{ki} = \mathcal{AE}(K_{EDApub_k}; K_{IP_i})$
5) Both y_i' and z_{ki} are forwarded to the next tool flow.

Fig. 1. Synplicity's Open IP Protocol [6].

would be highly beneficial to the system integrator but also to the IP vendor, by making his IP blocks available to parties that would otherwise be unable to pay large upfront fees. In the FPGA domain this is presently done in two different ways: (i) using proprietary encryption techniques from different EDA tool vendors and (ii) using encryption at the FPGA device level. However, proprietary solutions in general, assume that one system integrator has access to IP originating from a single source (in addition to the code/IP developed in house). This creates problems for IP integrators who combine IP from different IP vendors with their own IP using multiple EDA tools. Synplicity (an EDA vendor) has therefore suggested an open IP encryption scheme [6]. This would enable all EDA vendors to use a similar technique for IP encryption/decryption in their tools but with different keys.

A. The Typical Design Flow

Before introducing the encrypted design flow proposed in [6], it is necessary to understand what the typical hardware design flow (without encryption) for FPGA looks like. During the design process, a system integrator gets third party IP blocks from IP providers, typically in the form of a RTL, and combines them with its own (in house) RTL blocks. The design team then uses any one of the different EDA vendor's synthesis tool to generate a gate-level netlist. At this point, the design team might simulate their design and verify that the functionality as well as constraints from the design specifications are met. If this is not the case, then this first flow of designing and synthesizing is iterated several times until constraints and functionality are according to specification. Notice that synthesis and simulation tools might be provided by different EDA vendors. This (final) netlist is, in turn, provided to the FPGA vendor's place and route tools to generate a configuration file, which is programmed onto the FPGA.

B. Synplicity's Open IP Encryption Architecture

Figure 1 shows the overall scheme where an IP vendor encrypts his IP block using a symmetric cryptographic algorithm with a secret key (K_{IP}). This secret key is then encrypted using a public-key (asymmetric) cryptosystem using public keys for each EDA tool vendor. These encrypted keys are then embedded in the IP code. Therefore, the EDA tools (for which the IP vendor had encrypted the keys) first decrypts the key used to encrypt the IP block. This is possible as the secret key pair of the public key cryptosystem is embedded in the EDA tool. Once the IP vendor's key is decrypted, the tool can decrypt the IP block. Notice that this protocol allows the IP provider to decide ahead of time which tools will have access to his IP by encrypting the IP provider's key with the public key of each "authorized" EDA tool.

Although in theory the solution is secure, there are several drawbacks with the proposed solution, which seem evident to the security practitioner. The following is an explicit list of such problems or potential weaknesses with the proposed scheme. Notice that we are *not* claiming to break the system. Rather, we are attempting to be on the safe side and make sure that the proposed protocol is more robust against potential attackers:

- IP vendors have to completely trust the EDA vendors for the protection of their IP since the EDA vendor can decrypt any key used to decrypt an IP. This problem is further magnified with the proposed scheme as the IP vendor has to trust *all* the EDA tools used to process the IP block (here the same key is encrypted with public keys of multiple EDA vendors). Therefore, a weakness in *any* single tool of any vendor would be sufficient to compromise **all** the IP from all IP vendors.
- The encrypted IP block could be leaked by a licensee making it available to other unlicensed parties, who can perform all the above operations on the encrypted IP block and integrate the IP into a different design. The

main reason this is possible is that there is no binding of the encrypted IP to the licensed system integrator. In other words, anyone with a copy of the software tools can in principle use the flow and the IP block.
- In addition, during the flow if the entire IP output (combination from the different IP cores) has to be encrypted with only one of the many K_{IP_i} that was used as input IP blocks to the tool, then the output of the EDA tool can be compromised by choosing a known K_{IP_i} with a bogus IP.

C. Requirements and Assumptions

Given the previous discussion, one may ask what would we put forward as requirements of such a system? In this section, we briefly summarize these requirements from a security point of view. In particular, we would like to minimize the impact that a single key disclosure has on the system. In other words, if the secret key of an IP provider or the private key of an EDA tool is leaked to the outside, the effect that this disclosure has should be minimized in the overall system. Moreover, from an IP provider's point of view, it is desirable to have tools that could enforce a pay-per-use business model.

In our discussion, we assume implicitly that the environment in which the EDA tools operate is a secure environment or, alternatively, that the tools themselves use techniques to guarantee that the keys (and their use) present at any one time in memory are protected. We also assume that if the EDA tools contain legitimate and authentic secret keys then the tools are trusted. This is extremely important since if the keys are compromised (in particular the private key of the EDA tool vendor), then the overall security of the system falls apart. We cannot emphasize enough the importance of this last statement. Notice that our "software-based" solutions all suffer from this drawback. The way[1] that we find to get around this is to use the FPGA and link the IP directly to the FPGA. Assuming that the private keys of the EDA tools remain safe, we discuss in the next sections techniques to mitigate the exposure of the other keys used in the system.

Observe that in this model, an attacker who writes fake tools has no chance of success as long as the legitimate secret keys remain out of his reach. Similarly, we assume that only legitimate tool vendors will be able to get their keys certified. This certification can be performed either by a trusted third party designated by the industry or by the vendor itself by simply publishing its public keys on the EDA tool vendor's website, for example. Here it is implicitly assumed that the vendor's reputation and possibly its brand will act as the (implicit) certifying agent (i.e. by buying a tool from market leader AAA, we implicitly say that we trust its brand). In this work, the solutions presented make use of standard cryptographic techniques only. However, it is clear that these standard techniques can be combined with software obfuscation techniques or (for example) whitebox cryptography to improved the security of the system. We caution that such techniques are still an area of research in its infancy.

[1]Clearly, one way to guarantee the overall process to be secure is to require the EDA tools to run on trusted hardware. This, however, we think is not realistic. We do not explore in this work the possibility of relaxing this and allowing for only *part* of the computation to occur in trusted hardware.

IV. Strengthening The Open IP Flow

A. Using Secret-Sharing

Introduced independently by Shamir [20] and Blakley [21] in 1979, secret-sharing is a well-known technique to minimize the effects of key exposure. The idea is thus very simple. We follow the original flow but instead of re-encrypting part of the netlist with the key of the IP provider, we first split the output of EDA tool j into shares, such that the attacker has to compromise all keys before getting *any* information about the output of the EDA tool. Thus, this approach also strengthens the security at the "interface" between EDA tools. The overall flow is shown in Figure 2.

Notice that in Step 6, we could encrypt each K_{IP_i} separately. We assume that for the encryption of the secret keys K_{IP_i} of each IP provider, we have used a semantically secure encryption scheme [22]. Such schemes make use of nonces, which we do not show explicitly in the protocols for ease of presentation. Also in Fig.2, we have used the simplest form of secret sharing scheme. However, any secret-sharing scheme will work as well at the cost of additional hardware resources.

The proposed scheme as specified in the previous section results in an s fold increase in the communication complexity. In other words, if a normal netlist occupies 1 MByte of memory space, in the present scheme we will have to transmit s MBytes. To reduce this complexity, we can apply t-out-s secret sharing scheme where we only use t shares (instead of s). Then, only t shares are required and our memory requirements are reduced accordingly.

B. Using All-Or-Nothing Transforms

In [23] Rivest introduces a method intended to make it more difficult to perform exhaustive key searches on symmetric-key ciphers. Improvements to the efficiency of the All-Or-Nothing Transform (AONT) are known as well [24], which reduce the overhead from two encryptions to just one (i.e. the AONT would then require a few XOR operations together with an encryption operation). The AONT construction presented by Desai [24] is shown in Algorithm 1. We will denote the application of Algorithm 1 with input message N as $N' = AONT(N)$. Notice that once the AONT has been performed

Algorithm 1 The AONT from Desai [24].

Require: A message N split into blocks $N_1, N_2, N_3, \cdots, N_t$
Ensure: $t+1$ encrypted blocks $Y_1, Y_2, \cdots, Y_{t+1}$
1: Choose a random K'
2: **for** $i = 1$ to t **do**
3: $\quad N'_i = N_i \oplus AES(K', i)$
4: **end for**
5: $N'_{t+1} = K' \oplus N'_1 \oplus N'_2 \oplus \cdots \oplus N'_t$
6: **return** $N' = N'_1, N'_2, \cdots N'_{t+1}$

Assumptions and Notation:

- IP provider i has secret key K_{IP_i}
- Each EDA vendor (EDA$_j$) publishes a certified public key $K_{\text{EDA}pub_j}$ and the corresponding private key $K_{\text{EDA}priv_i}$ is embedded in a secure manner in the corresponding tool.
- The connection SYS-IPP is authenticated.

Secure Flow:

1) Transfer of IP block to EDA tools for processing

EDA Tool j — SYS — IPP

$y_i = \mathcal{SE}(K_{IP_i}; IP_i)$
$z_{ji} = \mathcal{AE}(K_{EDApub_j}; K_{IP_i})$

y_i, z_{ij} ← SYS ← y_i, z_{ij} ← IPP

$K_{IP_i} = \mathcal{AE}^{-1}(K_{EDApriv_j}; z_{ij})$
$IP_i = \mathcal{SE}^{-1}(K_{IP_i}; y_i)$

2) Process IP_i block and generate corresponding flow output N_j. The netlist might be divided into portions N_i corresponding to the original IP_i block. This, however, is not relevant in the new flow.
3) Assume that there are s different IP providers i.e. $i = 1, 2, 3, \cdots, s$. Then, the EDA tool generates $s-1$ random shares $R_2, R_3, R_4, \cdots, R_s$.
4) The EDA tool computes $N' = N \oplus R_2 \oplus R_3 \oplus R_4 \oplus \cdots \oplus R_s$.
5) The EDA tool encrypts N' and each of the R_i with each of the s K_{IP_i} to obtain $y'_1 = \mathcal{SE}(K_{IP_1}; N')$ and $y'_i = \mathcal{SE}(K_{IP_i}; R_i)$ for $i = 2, 3, \cdots, s$
6) EDA vendor tool j encrypts key(s) K_{IP_i} with the public-key of EDA vendor k, i.e. $z_k = \mathcal{AE}(K_{EDApub_k}; K_{IP_1}, K_{IP_2}, K_{IP_3}, \cdots, K_{IP_s})$
7) Both y'_i for $i = 1, \cdots, s$ and z_k are forward to the next tool flow.

Fig. 2. Strengthened Synplicity's Open IP Protocol with Secret Sharing.

on data N, the output N' is subsequently encrypted, resulting in $Y = \mathcal{SE}(K, N')$. The strength of the AONT idea is that an attacker must decrypt *all* the message blocks N'_i before he can obtain the entire N or any information on the parts N_i of N. In addition, the message expansion is only one extra encrypted block (i.e. in the case of AES only 128-bits extra). However, this is at the expense of performing two rounds of encryption. Integrating Algorithm 1 into the open IP flow then is straight forward. In particular, the following steps are performed:

1) Perform Steps 1 and 2 in Fig. 2. At the end of these steps we have output of EDA tool j, is N, which can be divided into blocks (of size specified by the block cipher to be used, i.e. 128-bit blocks if AES is used) N_i for $i = 1, 2, \cdots, t$.
2) The EDA tool generates *one* random key K'. This key does not need to be shared with anyone.
3) Compute $N' = AONT(N)$, where N' is the concatenation of blocks N'_i for $i = 1, 2, \cdots, t, t+1$.
4) The EDA tool encrypts $r = \lceil (t+1)/s \rceil$ blocks N'_i with each of the s keys K_{IP_i} to obtain $Y_i = \mathcal{SE}(K_{IP_i}; N'_{(i-1)s+1}, N'_{(i-1)s+2}, \cdots N'_{(i-1)s+s})$ for $i = 1, 2, \cdots, s$. In other words, we partition the $t+1$ blocks N'_j into s groups each containing r blocks, except possibly the last group. Each group of r blocks is then encrypted with a different key K_{IP_i}.
5) EDA vendor j's tool encrypts key(s) K_{IP_i} with the public-key of EDA vendor k, i.e. $z_k = \mathcal{AE}(K_{EDApub_k}; K_{IP_1}, K_{IP_2}, K_{IP_3}, \cdots, K_{IP_s})$
6) All Y_i and z_k are forwarded to the next tool flow.

The overhead of two encryption operations (instead of only one) seems a reasonable trade-off considering that encryption will only be performed at the beginning and at the end of each tool flow.

C. Discussion

The solutions so far strengthen the open IP flow by making it harder for an attacker to extract information from the intermediate outputs sent between one tool flow and the next. The first solution does this by using secret-sharing on the output of one tool and re-encrypting with the keys originally provided by the IPP. The cost is an increase in the communication complexity since now, the EDA tool needs to encrypt s outputs (or a subset of it), where s is the number of IP blocks originating from different IP providers[2]. The second solution, based on AONTs, has similar properties to the first one. It reduces the communication complexity to a fraction of the output of each EDA tool at the cost of increased computational complexity. In other words, instead of performing one round of encryption per block of the EDA tool output, we need to perform two rounds.

The previous protocols can be strengthened to minimize the effect of key leakage between different EDA tools, by having each EDA tool generate new keys, call them $\{K_{Re-encrypt_i}\}_{i=1}^{s}$, and re-encrypt their outputs with these keys instead of re-using $\{K_{IP_i}\}_{i=1}^{s}$. Notice that in this case, the system needs to take care of encrypting these keys with public keys approved by the IPP in advance[3].

Although, the solutions described in this section reduce the effect of key leakage between EDA tools, drawbacks remain. In particular, if an IP block is leaked (in encrypted form) to the outside world anyone with a legal set of design tools

[2]Here we have made the implicit assumption that IP blocks corresponding to the same IP provider have been encrypted with the same key. Clearly, the proposed system can support different IP blocks encrypted with different keys developed by the same IP provider. Simply treat each IP block independently as being developed by a different provider.

[3]In the original scheme [6], the IPP encrypts his key with the public keys of those tools, which he allows to process his code. Notice that this is done in advance, thus, essentially providing an implicit license.

(which have knowledge of the private key of the EDA vendor) will be able to use the encrypted block. It appears hard to completely solve this problem unless you link the IP block to the specific hardware platform, e.g. using PUFs as in [1], [4], [5] or modifying the hardware platform as in [7]. What we can do is reduce the ability of an attacker to use an IP block in multiple EDA tool flows owned by different organizations. In the following we explore possible solutions to this problem.

V. Public-Key Cryptography Based Variants

In this section, we consider ways in which we can limit the distribution of encrypted IP. In particular, notice that a dishonest system integrator does not necessarily need to get access to the IP block key K_{IP_i} (corresponding to IP provider i) to be able to integrate this IP into his design. In particular, given that the K_{IP_i} has been encrypted with the public key K_{EDApub_j} of EDA tool j, then any tool with the corresponding private key $K_{EDApriv_j}$, will have access to K_{IP_i}. This implies that one system integrator can provide the encrypted IP block to another system integrator with the same tools and the second system integrator will be able to use the IP block, since he has a legal copy of the EDA tool j.

Several possible solutions come to mind. One could try an approach in which the IPP uses a different K_{IP_i} for every SYS. This will clearly *not* solve the problem because of the same reasons described in the previous paragraph. Using a different public-key/private-key pair for each EDA tool provided to each SYS would solve the problem but it would essentially be an "almost" symmetric-key cryptosystem. In particular, we loose the advantage of having a single publicly available and global public key K_{EDApub_j}, which can be used by everyone desiring to encrypt a message to the owner of the corresponding private key $K_{EDApriv_j}$. The EDA tool vendor would have to make available the K_{EDApub_j} to the IPP , in a secure and authenticated manner, every time that the IPP desires to provide its IP blocks to a different SYS. In doing this the EDA vendor would act as an intermediary, which might not be desirable from a security point of view but also an economic/strategic point of view. For example, the EDA vendor would know the IPP customer lists, which might not be acceptable to the IPP. Fortunately, standard cryptographic techniques can be used to solve this problem. Figure 3 shows the initial part of the protocol, which is critical for the security of the overall protocol. Once the key used to encrypt the IP block has been derived, one of the protocols previously described in Sect.IV can be used. Thus, we follow ideas similar to [7], [8] but this time at the EDA tool level, which has the advantage of not requiring changes to the hardware platform (FPGA chip).

In Figure 3 the most involved part is the authenticated key exchange protocol, which is due to Diffie et al. [25] and it is known as the Station-To-Station protocol. Notice that we just chose a well-known and studied protocol but other protocols, which achieve the same exist as well (see e.g. [26]). Observe also that we use a Key Derivation Function, denoted KDF, which is not defined. This is left on purpose this way as there is currently work to standardize methods to do this (see [27], [28] for proposals). The basic method, however, essentially derives the session key by computing several hashes/MACs of the Diffie-Hellman session key. To summarize, what Figure 3 shows is how to perform an authenticated key exchange and subsequently, use the agreed key to encrypt the IP block. Implicitly we make the assumption that the IPP performs this key exchange once with the first tool that his IP block will interact with and at the same time he certifies the public keys of the tools to which his IP can be forwarded.

Notice that this solution partially solves the problem of a licensee leaking an IP block and making it available in encrypted form to other (legal) EDA tool licensees. In particular, we create a key which is specific to a particular EDA tool instantiation (by instantiation we mean the particular piece of software that the SYS gets from the EDA tool provider) using the KDF and assuming that there is a unique identifier which cannot be removed or tampered with in the EDA tool. Such unique identifier could be implemented with tamper resistance tokens or a TPM, for example. A protocol would have to run between the tool and the token to guarantee that the unique identifier is indeed the expected value. Assuming that the attacker cannot tamper with the identifier then our protocol guarantees that every EDA Tool instantiation will have access to a different key and therefore, leaking the IP block in encrypted format would not be of use, since other tools will not know the agreed key.

VI. Secure Integration of Multiple IP-Blocks Based On PUFs

As mentioned in Sect. III, during the integration phase, the various IP blocks undergo different design phases. Firstly they are simulated to ensure they fulfill the required criteria. Then they are integrated into a single design by synthesizing, place and route tools.

In this section, we sketch a solution requiring that the IP vendors provide a simulation library unencrypted for their IP which allows the integrator to test the IP block with other IP (and his own design). Such a simulation library is already commonly distributed during the evaluation phase in the industry and it does not allow to obtain a synthesizable core from it. The real IP that goes onto the FPGA is sent to the SYS as an encrypted pre-routed hard macro. Such macros are also already in use today (for e.g. Altera's incremental compilation). The difference here is that the key for the decryption is not available to the tools but only inside the FPGA (and hence would require modification to the configuration controller of the FPGA which will be described later). The macro is thus encrypted using a key derived from the PUF responses during the enrollment phase of the FPGA. For specific protocols, describing how to perform the enrollment and how to integrate it with symmetric key and public-key primitives in the context of IP protection for FPGAs, we refer to [4], [5], respectively. Notice that the public-key based protocols described in [5] do not require that any secret information leave the FPGA, which is an added advantage. It is only required that the public key

Assumptions and Notation:

- IP provider i has a private-public key pair $(K_{IPpriv_i}, K_{IPpub_i})$.
- Each EDA vendor (EDA$_j$) publishes a certified public key $K_{\text{EDA}pub_j}$ and the corresponding private key $K_{\text{EDA}priv_i}$ is embedded in a secure manner in the corresponding tool. Thus, each IP provider i has a certified copy of the EDA tool vendor's.
- The EDA vendor has provided IP provider i with a signed certificate on his public key $\text{Cert}(\text{Info}_{IP_i}) = (\text{Info}_{IP_i}, \mathcal{S}ig(K_{EDApriv_j}; \text{Info}_{IP_i}))$, where $\text{Info}_{IP_i} = (ID_{IP_i}, K_{IPpub_i})$
- A Key Derivation Function, denoted KDF.

Secure Key Exchange [25]:

EDA Tool j		SYS		IPP
				Generate random y
		$\xleftarrow{K_{A_i}}$		$K_{A_i} = g^y \mod p$
Generate random x				
$K_{B_j} = g^x \mod p$				
$K_{ij} = KDF(K_{A_i}, K_{B_j}, ID_{IP_i}, ID_{EDA_j})$				
$Y_{BA_{ji}} = \mathcal{SE}(K_{ij}; \mathcal{S}ig(K_{\text{EDA}priv_j}; K_{B_j}, K_{A_i}))$				
		$\xrightarrow{K_{B_j}, Y_{AB_{ji}}}$		
				$K_{ij} = KDF(K_{A_i}, K_{B_j}, ID_{IP_i}, ID_{EDA_j})$
				$T' = \mathcal{SE}^{-1}(K_{ij}; Y_{BA_{ji}})$
				$\mathcal{V}er(K_{\text{EDA}pub_j}; T')$. If correct continue, otherwise abort
				$Y_{AB_{ij}} = \mathcal{SE}(K_{ij}; \mathcal{S}ig(K_{IPpriv_i}; K_{A_i}, K_{B_j}))$
		$\xleftarrow{Y_{AB_{ij}}}$		
$T'' = \mathcal{SE}^{-1}(K_{ij}; Y_{AB_{ij}})$				
$\mathcal{V}er(K_{\text{IP}pub_i}; T'')$				
If correct continue, otherwise abort.				

IP Block Transmission:

		$\xleftarrow{Y_i}$		$Y_i = \mathcal{SE}(K_{ij}; IP_i)$
$IP_i = \mathcal{SE}^{-1}(K_{ij}; Y_i)$				

Fig. 3. Public-key Based IP Sharing Shared Key Protocol.

corresponding to the private key derived from the PUF and stored in the FPGA be certified by a trusted authority, which could be the FPGA manufacturer.

Once the designer is sure that the IP works in the overall design (based on the simulation results), the integrator can position the black box encrypted IP at a particular position on his FPGA. The interfacing of this block is made possible by the exact position of the interface signal pins made available by the IP provider. Since the encrypted black boxes are pre-routed for a particular FPGA, the final bitfile that is used to program the FPGA can set aside these resources on the FPGA. Thus, the bitfile contains tags for which IP block (when decrypted) goes to which empty position and the required helper data position in external flash to decrypt the block. The overall programming file for the FPGA then includes this bitfile, the encrypted IP blocks and their associated helper data.

By using the techniques described, we can ensure that the IP vendor has an end-to-end security of his IP. The details of the IP are never decrypted in any EDA tool. In particular, decryption only occurs inside the FPGA. The secret key for the decryption is also unknown to any other user (including the system integrator) because it is based on the PUF responses which are specific to the FPGA.

As mentioned before the configuration controller on the FPGA also needs to be modified to take care of partially programming parts of the FPGA (this is already possible for some Xilinx FPGAs but under different circumstances). The modified configuration controller (CC) functions in the following way:

1) The CC first loads the bitfile and programs areas of the FPGA which can be done without any decryption, leaving out the empty blocks for the decrypted IP.
2) The CC then reads the tags to program the empty areas with the appropriate IP block.
3) Based on the tag information it then reads the helper data for the encrypted block and based on the PUF response of the FPGA to derive the necessary key to decrypt the IP.
4) The CC then decrypts the IP block and programs the empty positions.
5) CC continues to perform (3) and (4) until all encrypted IP blocks are programmed onto the FPGA.

The keys that are used to encrypt different IP blocks from different IP vendors can be kept different (even though they are for the same FPGA). It is imperative that the keys be derived from different PUF blocks if traditional techniques for fuzzy extractors [17], [16] are used. In particular, there are recent attacks [29] which allow to derive the original PUF key based on the availability of different helper data (derived from the same randomness). The algorithmic circuitry to derive the keys from the PUF and the helper data can be built-in by the FPGA manufacturers onto the configuration controller. A

single crypto decryption can be used if all IP vendors decide to choose a standardized algorithm and this can also be included within the configuration controller (which is already being done for high end FPGAs).

VII. Conclusions

IP protection for FPGA cores have previously considered a single IP owner model. However in a real FPGA development environment, multiple IP providers are involved whose cores are part of the final design. Hence EDA tools play a central role in any IP protection framework. The Synplicity's Open IP Protocol is a first step in the direction of creating an open environment for different tools to inter-operate with each other while protecting the IP from various vendors. Our contribution has shown possible weaknesses in the proposed protocol and suggested modifications that allow robust protection of IP. We have also presented a solution for IP rationing (i.e. limit the use) using PUFs. The proposed solution is extremely important to guarantee per use royalty for IP when multiple IP from different vendors needs to be integrated into a single design. This solution also makes sure that the confidentiality of the IP is in one single chain rather than having to trust EDA tool vendors and possible weakness in a parallel chain.

References

[1] E. Simpson and P. Schaumont, "Offline Hardware/Software Authentication for Reconfigurable Platforms," in *Cryptographic Hardware and Embedded Systems — CHES 2006*, ser. LNCS, L. Goubin and M. Matsui, Eds., vol. 4249. Springer, October 10-13, 2006, pp. 311–323.

[2] J. Lach, W. H. Mangione-Smith, and M. Potkonjak, "Fingerprinting Digital Circuits on Programmable Hardware," in *International Workshop on Information Hiding*, ser. LNCS, D. Aucsmith, Ed., vol. 1525. Springer, April 14-17, 1998, pp. 16–31.

[3] T. Kean, "Cryptographic rights management of FPGA intellectual property cores," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays — FPGA 2002*, 2002, pp. 113–118.

[4] J. Guajardo, S. S. Kumar, G.-J. Schrijen, and P. Tuyls, "FPGA Intrinsic PUFs and Their Use for IP Protection," in *Cryptographic Hardware and Embedded Systems — CHES 2007*, ser. LNCS, P. Paillier and I. Verbauwhede, Eds., vol. 4727. Springer, September 10-13, 2007, pp. 63–80.

[5] ——, "Physical Unclonable Functions and Public Key Crypto for FPGA IP Protection," in *International Conference on Field Programmable Logic and Applications — FPL 2007*. IEEE, August 27-30, 2007, pp. 189–195.

[6] A. Dauman, "An Open IP Encryption Flow Permits Industry-Wide Interoperability," Synopsys, Inc., White Paper, June 2006, available at http://www.synplicity.com/literature/whitepapers/.

[7] S. Drimer, T. Güneysu, M. G. Kuhn, and C. Paar, "Protecting multiple cores in a single FPGA design," Draft available at http://www.cl.cam.ac.uk/ sd410/, written May 2008. Available on-line August 2008.

[8] T. Guneysu, B. Moller, and C. Paar, "Dynamic Intellectual Property Protection for Reconfigurable Devices," in *Field-Programmable Technology, 2007. ICFPT 2007. International Conference on*, 2007, pp. 169–176.

[9] W. Diffie and M. E. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory*, vol. IT-22, no. November, pp. 644–654, 1976.

[10] R. S. Pappu, "Physical one-way functions," Ph.D. dissertation, Massachusetts Institute of Technology, March 2001.

[11] R. S. Pappu, B. Recht, J. Taylor, and N. Gershenfeld, "Physical one-way functions," *Science*, vol. 297, no. 6, pp. 2026–2030, 2002.

[12] B. Gassend, D. E. Clarke, M. van Dijk, and S. Devadas, "Silicon physical unknown functions," in *ACM Conference on Computer and Communications Security — CCS 2002*, V. Atluri, Ed. ACM, November 2002, pp. 148–160.

[13] P. Tuyls, G.-J. Schrijen, B. Škorić, J. van Geloven, N. Verhaegh, and R. Wolters, "Read-Proof Hardware from Protective Coatings," in *Cryptographic Hardware and Embedded Systems — CHES 2006*, ser. LNCS, L. Goubin and M. Matsui, Eds., vol. 4249. Springer, October 10-13, 2006, pp. 369–383.

[14] S. S. Kumar, J. Guajardo, R. Maes, G.-J. Schrijen, and P. Tuyls, "The Butterfly PUF: Protecting IP on every FPGA," in *IEEE International Workshop on Hardware-Oriented Security and Trust — HOST 2008*, M. Tehranipoor and J. Plusquellic, Eds. IEEE Computer Society, June 9, 2008, pp. 67–70.

[15] E. Öztürk, G. Hammouri, and B. Sunar, "Physical unclonable function with tristate buffers," in *International Symposium on Circuits and Systems (ISCAS 2008)*. IEEE, May 18-21, 2008, pp. 3194–3197.

[16] Y. Dodis, M. Reyzin, and A. Smith, "Fuzzy extractors: How to generate strong keys from biometrics and other noisy data," in *Advances in Cryptology —- EUROCRYPT 2004*, ser. LNCS, C. Cachin and J. Camenisch, Eds., vol. 3027. Springer-Verlag, 2004, pp. 523–540.

[17] J.-P. M. G. Linnartz and P. Tuyls, "New Shielding Functions to Enhance Privacy and Prevent Misuse of Biometric Templates," in *Audio-and Video-Based Biometrie Person Authentication — AVBPA 2003*, ser. LNCS, J. Kittler and M. S. Nixon, Eds., vol. 2688. Springer, June 9-11, 2003, pp. 393–402.

[18] C. Bösch, J. Guajardo, A.-R. Sadeghi, J. Shokrollahi, and P. Tuyls, "Efficient Helper Data Key Extractor on FPGAs," in *Cryptographic Hardware and Embedded Systems — CHES 2008*, ser. LNCS, E. Oswald and P. Rohatgi, Eds., vol. 5154. Springer, August 10-13, 2008, pp. 181–197.

[19] A. B. Kahng, J. Lach, W. H. Mangione-Smith, S. Mantik, I. L. Markov, M. Potkonjak, P. Tucker, H. Wang, and G. Wolfe, "Watermarking techniques for intellectual property protection," in *Design Automation Conference — DAC '98*. New York, NY, USA: ACM Press, 1998, pp. 776–781.

[20] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[21] G. R. Blakley, "Safeguarding cryptographic keys," in *Proceedings of the National Computer Conference*, vol. 48, 1979, pp. 313–317.

[22] S. Goldwasser and S. Micali, "Probabilistic Encryption," *J. Comput. Syst. Sci.*, vol. 28, no. 2, pp. 270–299, 1984.

[23] R. L. Rivest, "All-or-Nothing Encryption and the Package Transform," in *Fast Software Encryption — FSE '97*, ser. LNCS, E. Biham, Ed., vol. 1267. Springer, January 20-22, 1997, pp. 210–218.

[24] A. Desai, "The Security of All-or-Nothing Encryption: Protecting against Exhaustive Key Search," in *Advances in Cryptology — CRYPTO 2000*, ser. LNCS, M. Bellare, Ed., vol. 1880. Springer, August 20-24, 2000, pp. 359–375.

[25] W. Diffie, P. C. van Oorschot, and M. J. Wiener, "Authentication and Authenticated Key Exchanges," *Des. Codes Cryptography*, vol. 2, no. 2, pp. 107–125, 1992.

[26] H. Krawczyk, "SKEME: A versatile secure key exchange mechanism for the Internet," in *Proceedings of the Symposium on Network and Distributed Systems Security (NDSS '96)*. Internet Society, February, 1996, pp. 114–127.

[27] E. Barker, D. Johnson, and M. Smid, "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography (Revised)," National Institute of Standards and Technology, NIST Special Publication SP 800-56A, March 2007, available at http://csrc.nist.gov/publications/PubsSPs.html.

[28] H. Krawczyk, "On Extract-then-Expand Key Derivation Functions and an HMAC-based KDF," Draft available at http://www.ee.technion.ac.il/ hugo/kdf/, March 10, 2008.

[29] K. Simoens, P. Tuyls, and B. Preneel, "Privacy weaknesses in biometric sketches," in *IEEE Security and Privacy Symposium*. IEEE Computer Society, 2009, p. To appear.