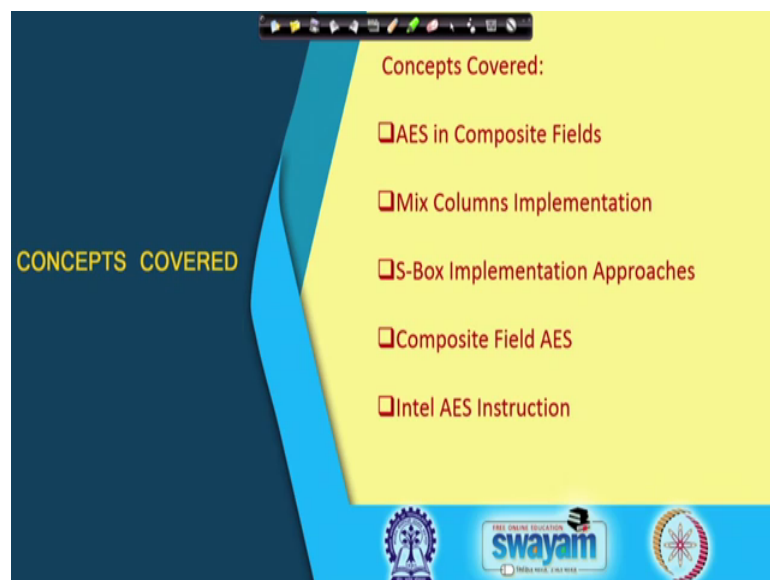


Hardware Security
Prof. Debdeep Mukhopadhyay
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 14
Hardware Implementation of Advanced Encryption (Contd.)

So, welcome to this class on Hardware Security. So, we shall be discussing on what we were essentially we shall be continuing on what we were discussing in the last class on various hardware techniques for implementing advanced encryption standards.

(Refer Slide Time: 00:35)



So, so to start with today we shall be discussing in details about how we can express the entire AES in composite field something that we are already discussing in the last class, we shall continue with that. In particular, we shall be looking into how a mix column is implemented, along with an S-box or various approaches for implementing S-boxes.

So, and finally right we shall be looking into how we can design, and we shall be also looking into how we can design a combined structure of between the encryption unit and the decryption unit that is how we can combine the encryptor and the decryptor of AES. And finally, we take a look at one of the very new are you know like nearly new implementations on AES state of the art implementation of AES, which is integrated in most of our modern Intel processors.

(Refer Slide Time: 01:21)

The slide is titled "SubBytes in Composite Fields" and contains the following content:

- $Y = AX^{-1} + B$.
- In composite fields, $Y' = T(Y) = T(AX^{-1}) + TB = TAT^{-1}(T(X^{-1})) + TB = A'X' + B'$, where the affine matrixes A and B are updated as:
$$A' = TAT^{-1}, B' = TB$$

A note in a box states: "It may be emphasized that due to the isomorphism property, $T(X^{-1}) = (T(X))^{-1}$. Thus, we apply the transformed matrices A' and B' on the data which is in the composite field. Thus we can avoid the intermediate field transformations."

At the bottom of the slide, there are logos for "THE ONLINE EDUCATION swayam" and "INDIA WISE" along with a circular emblem.

So, to with this background right let us take a look and to what we have to discuss. So, to start with, we have already discussed this in the last class that suppose let us consider the AES sub byte operation. So, this is the essential steps that we do in the AES sub byte, we take a finite field inverse, we do an affine transformation that means, a pre-multiply with a fixed matrix A followed with by adding the vector B.

So, now one approach could be that we do this in composite fields. But, when we do this in composite field, there is an always and conversion that we do from the standard field to the composite field and back. So, this will incur an overhead. So, one approach that is essentially advised is that let us transform the entire AES in composite fields, so that at the very beginning wherever I have the raw plaintext and the key, there we do a transformation into the composite field.

We do the entire AES in one of in the composite field not only the only the sub bytes, but also the other operations like meets columns and things like that. And finally, right we get the result that is the cipher text in the composite field, we bring back the result to its normal field by taking the inverse mapping or applying the inverse mapping.

So, in this case let us take a look about that is about you know like suppose it is not only that the inverse has been done in composite field, what if I do the entire sub byte operation in composite field that means, along with the affine mapping. In that case,

what we have to do is that we have to slightly adjust the matrices A and B, if you want to do the operation in composite field. So, how do we do that is illustrated here.

So, if I apply the transformation from GF 2 to the power of 8 to the composite field, say GF 2 to the power of 4 square by the mapping T or the matrix T. Then essentially; we know that $Y \text{ dash} = T Y$, so we apply T on both sides of this equation. So, we get $T A X \text{ inverse} + T B$ that is T has been applied to both A X inverse as well as B.

Now, we can easily write $T A X \text{ inverse}$ as $T A T \text{ inverse}$ multiplied with $T X \text{ inverse}$, so note that $T \text{ inverse}$ with T is I or the identity matrix and plus T B. So, this $T A T \text{ inverse}$ is what I call as A dash, and this $T X \text{ inverse}$ or T applied on X inverse is what is called as X dash ok, and T B has been denoted as B dash. So, therefore these are my transformed affine matrices A and B ok. So, they are been updated by $T A T \text{ inverse}$ and B dash has been updated as T B ok.

So, an important point which may be emphasized here is that because of isomorphism property note that what we are doing right here is important to observe. So, what we are doing here is we are basically denoting this $T X \text{ inverse}$ as X dash right, so as if this is the inverse in the composite fields. So, note that it is correct why because of the isomorphism property one can calculate X inverse in the original field, and then transformed by apply applying T, it can transform it into the composite field or one can also apply or convert X directly into the composite field, and then take its inverse right.

So, therefore this is an identity in because of the isomorphism property, so that implies that I can substitute these $T X \text{ inverse}$ as $T X \text{ whole inverse}$, so that means what that means, I am doing the inverse in the composite field X was originally in the GF 2 to the power was in GF 2 to the power of 8 by applying T X, I have transformed it into the composite field.

And now this operation right is done in the composite field ok, so that is why this equation that is $Y \text{ dash} = A \text{ dash} X \text{ dash} + B \text{ dash}$ is in the composite field that means, X dash is in the composite field, A dash and B dash also has been adjusted to do the transformation in the composite field without requiring this transformation from the GF 2 power of 8 to GF 2 power of 4 squared and vice versa at every step ok, so that is why you have you now have a consolidated sub bytes which where this entire sub bytes

has been expressed in composite fields ok, so that will reduce your overheads incurred due to field transformations.

(Refer Slide Time: 05:37)

The slide contains the following text:

- The irreducible polynomial used is say $Y^2 + Y + \tau$. Here τ is a primitive element of $GF(2^4)$. We choose, $\tau = \omega^{14}$, where $\omega = (0010)_2$ is an element in $GF(2^4)$.
- A key point is that depending on the choice above the matrix A' will change, and also its cost.
- For example, with the above choice let us compare with the original matrix for AES. The number of 1s reduce from 40 to 18!!!

The slide shows two matrices, A and A' , with an arrow pointing from A to A' . Matrix A is:

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Matrix A' is:

$$A' = \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

The slide also features logos for Swamyam and other educational institutions, and a small video inset of a presenter in the bottom right corner.

So, now there are some more interesting observations here also. Like here is an example, so the irreducible polynomial that we choose here, so that is as we have discussed in the prior classes as well. We have got say $Y^2 + Y + \tau$, where τ is a primitive element of $GF(2^4)$ ok.

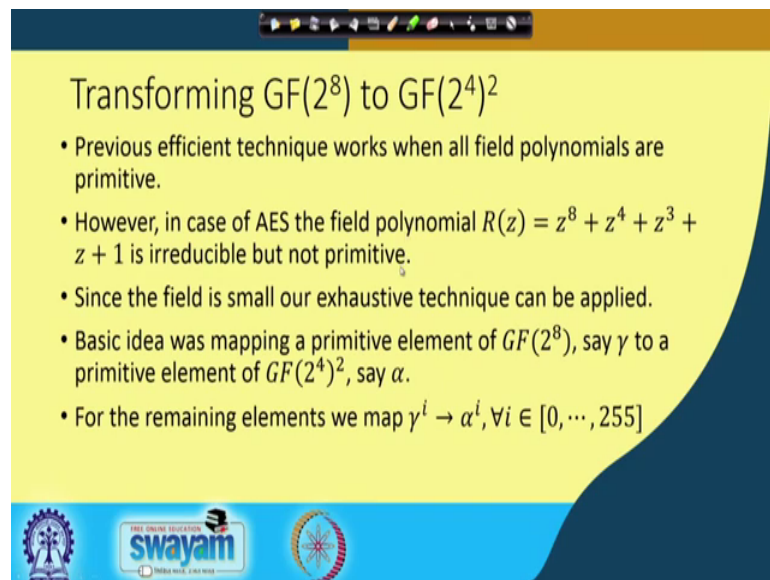
So, we choose so this is an example of τ that we choose say τ is ω^{14} , where ω is X that is 0010. So, this is like just an element in $GF(2^4)$. And we raise it to the power of 2 power of 14 to serve as a value of τ ok. So, you can have different such values of such values or such (Refer Time: 06:13) candidates that you can choose, but it is important to note that τ the τ is a primitive element of $GF(2^4)$ ok. So, as we know that there are different primitive elements of $GF(2^4)$, so you can actually play around with those values. And you can you know like if you if you choose different values of τ , one you can easily understand that you will get different choices for the matrix A A dash also ok.

So, in this case for example, a key point is that depending on the choice of the matrix A dash will change. And therefore, it will have different 01 values. Like for example, in this case the value of or the matrix A has been transformed into A dash, and it looks like

this ok. So, you can observe that the number of ones in the original matrix of A is significantly large compared to the number of ones, which we have in A dash ok.

And therefore, right it is an interesting choice, because in this case the number of ones is around 40, and here it is around 18. So, therefore the number of gates or the number of XOR's which you will require to implement this transformation will be much less, when you are doing the operation in the composite fields ok, when you are doing it in composite fields. So, therefore this naturally gives a nice mechanism of how you can reduce the cost of your implementations ok. So, you can try to play around with different choices, and you can try to see where you get exactly minimal number of 1s for example in A dash ok, so it could be an interesting exercise in itself.

(Refer Slide Time: 07:39)



The slide is titled "Transforming $GF(2^8)$ to $GF(2^4)^2$ ". It contains the following bullet points:

- Previous efficient technique works when all field polynomials are primitive.
- However, in case of AES the field polynomial $R(z) = z^8 + z^4 + z^3 + z + 1$ is irreducible but not primitive.
- Since the field is small our exhaustive technique can be applied.
- Basic idea was mapping a primitive element of $GF(2^8)$, say γ to a primitive element of $GF(2^4)^2$, say α .
- For the remaining elements we map $\gamma^i \rightarrow \alpha^i, \forall i \in [0, \dots, 255]$

At the bottom of the slide, there are logos for "THE INDIAN EDUCATION SWAYAM" and "INDIA RISE, EDUCATION THRIVES".

So, so therefore right I mean we would like to now transform, so let us take I mean concrete example. So, we had supposed transforming $GF(2^8)$ to $GF(2^4)^2$. And we have already discussed about pre efficient techniques, and techniques about how we can do these transformations previously. But, note that in the efficient technique that we discussed right in the last class or in one of my last classes they the polynomial so we need a mapping of the basis vectors ok.

So, in that case right the field polynomials were assumed to be a primitive polynomial. But, note that in case of AES the polynomial $R(z)$, which is equal to $z^8 + z^4 + z^3 + z + 1$ is irreducible, but is not primitive ok. So, we

do not directly apply that technique, but rather since the field is small we can actually do our exhaustive analysis ok. So, what we can do is that, we can take the primitive element of GF 2 power of 8 say gamma, and we can map it to a primitive element of GF 2 power of 4 square say alpha ok.

So, what we can do is for all the values from 0 to 255 that means for all the 256 values I just do this mapping gamma power of i 2 alpha power of i, and get a mapping ok. So, it is interesting to check that whether the multiplicative homomorphism, and the additive homomorphism are satisfied for these choices ok.

(Refer Slide Time: 09:01)

An Example T from $GF(2^8)$ to $GF(2^4)^2$

$$T = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \end{pmatrix}$$

Consider $2 \in GF(2^8)$, ie can be denoted as (0000 0010).
Thus, $T(2)$ would be in $GF(2^4)^2$, ie. (0010 1110) = $\alpha x + \alpha^{11}$, where α is the primitive element of $GF(2^4)$, modulo $x^4 + x + 1$.

Atri Rudra, Pradeep K. Dubey, Charanjit S. Jutla, Vijay Kumar, Josyula R. Rao, Pankaj Rohatgi: Efficient Rijndael Encryption Implementation with Composite Field Arithmetic. CHES 2001: 171-184

So, you know like suppose we get like a nice I mean it is done like we get this mapping, and it and it satisfies the multiplicative and the homomorphic property. Then, this is an candidate matrix ok. So, you can act again you know like and get different matrices, but this matrix is say our choice ok. So, you can get more illustrations about this matrix is in this paper, which was published in CHES in 2001.

So, so in this case right if you take this mapping, there are certain interesting points that you can observe is that the last column for example in this matrix is one, which means like in unity element in GF 2 power of 8, we will get mapped into GF 2 power of 4 square also as a unity element ok. So, likewise right you can also observe that is 0 will get mapped into 0, because that is obvious. But, for the other parts right it is interesting that to see how they work ok.

So, I give you an example here of how the mapping looks for or non-trivial input. So, for example, two right is an element in GF 2 to the power of 8, suppose the two is an field element, you can also write two in curly braces to indicate that it is not two per say. So, you can actually you know like denote these two as this value say 0000 and 0010 that stands for x basically.

So, therefore T 2 would be in now GF 2 to the power of 4 square. So, if I apply this matrix on two, then I should get the result in my composite field GF 2 power of 4 square. So, this turns out to be this column here that is this column, because you note that there is only 1 1 here ok. So, therefore this turns out to be this column, and that is 0 0 1 0 followed by triple 1 0 ok. So, now note that this element is in the composite field. So, how do we read this, so this part is my top part that is this part has got I mean is the coefficient of x ok, whereas this part is the constant part ok.

So, therefore this part has a as that the coefficient value here is alpha, because here the coefficient is itself you know it can be denoted as alpha, where alpha is a primitive element of GF 2 to the power of 4 ok. So, you can also say that in GF 2 to the power of 4 this is again x ok. We are there are you are using four four bits to denote the value ok.

So, likewise this value 1 1 1 0, so I leave it to an exercise to verify that you can take alpha. And if you raise it to the power of 11, and do a modulo right which is x power of 4 plus x plus 1, which is an irreducible polynomial in GF 2 power of 4, then you will get alpha power of 11 ok. And if you get alpha power 11, then this alpha power 11 stands for this vector that is in this in the field GF 2 to the power of 4, this is x cubed plus x square plus x ok, so you should get that value. So, if you do that right in a compact way, you can denote them as alpha x plus alpha power of 11 ok. So, now what you can what the reason why I show in particular for 2 is will be understood, if you consider now the mixed column matrix ok.

(Refer Slide Time: 11:59)

Mix Columns in Composite Fields

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix}$$
In Composite Fields

$$\begin{bmatrix} T(2) & T(3) & T(1) & T(1) \\ T(1) & T(2) & T(3) & T(1) \\ T(1) & T(1) & T(2) & T(3) \\ T(3) & T(1) & T(1) & T(2) \end{bmatrix}$$

Consider the composite field $\text{GF}((2^4)^2)$ with irreducible polynomial $x^2 + x + \alpha^{14}$. Every element in this field is represented as $\mathbf{a_1x + a_0}$, where $\mathbf{a_1}$ and $\mathbf{a_0} \in \text{GF}(2^4)$.

$$T(1) \cdot (a_1x + a_0) = (a_1x + a_0)$$

$$T(2) \cdot (a_1x + a_0) = x[(\alpha(a_1 + a_0)) + a_1\alpha^{11}] + [a_1 + \alpha^{11}a_0]$$

$$T(3) \cdot (a_1x + a_0) = x[(\alpha(a_1 + a_0)) + a_1\alpha^{11} + a_1] + [a_1 + \alpha^{11}a_0 + a_0]$$

$$= T(2) \cdot (a_1x + a_0) + (a_1x + a_0)$$

So, remember that in the mix column matrix, we have got these elements. And as I said that I now I want to express the entire AES in composite fields, it means that I have to also transform these elements into the composite fields ok. So, these are constant elements. They need to be mapped into say T 2, T 3, T 1 and so on. Of course T 1 will be 1, but T 2 and T 3 will be non-trivial. So, you have to basically calculate it ok.

So, therefore write T 1 is T 4 so remember that when I am doing this operation, then my elements that I am multiplying them are now not in GF 2 power of 8, but in GF 2 power of 4 square. So, therefore they will be they will be represented as say a 1 x plus a naught, so if I multiply with T 1, I will get a 1 x plus a naught, because T 1 is one itself. For T 2 right T 2 when I multiply with a 1 x plus a naught, so note that T 2 right essentially is as we have found out in the previous discussion, it is alpha x plus alpha to the power of 11. So, therefore I will take now alpha x to the power of 11, and alpha x plus alpha to the power of 11, and multiply it with a 1 x plus a naught ok.

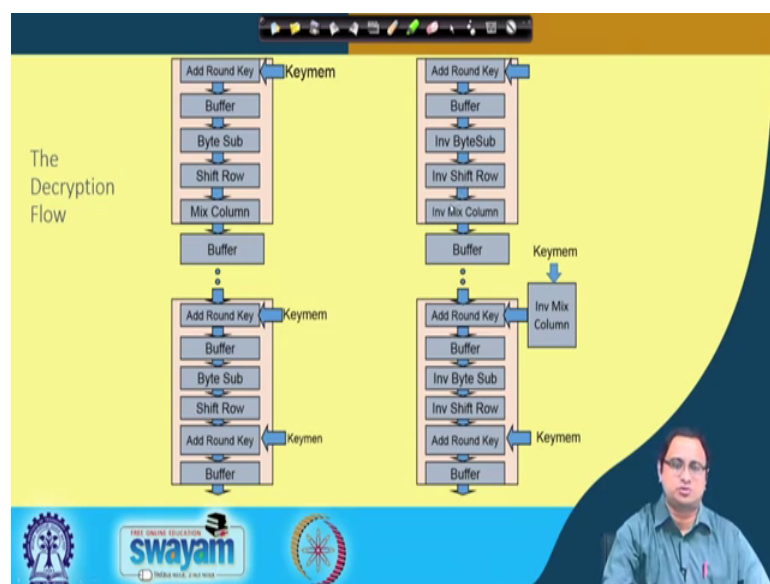
So, again I leave it to as an exercise to check that this equation will be satisfied. You can observe that when there will be instances, when you get x squared, but that x square needs to be you know like converted back into the field GF 2 power of 4 squared by taking an XOR or you know like by considering a modulo with this polynomial, which is an irreducible polynomial of the composite field GF 2 power of 4 square.

So, what is the polynomial $x^2 + x + \alpha^{14}$. So, therefore whenever you get x^2 , you substitute that with $x + \alpha^{14}$. So, if you do that right every element here can be represented in this form of a $1x + a$, where a 1 and a naught are elements in $GF(2^4)$.

So, in this case right you will get this particular example here. So, therefore you can observe that you are essentially doing more operations in the sense like you are doing additions, but note that the multiplications that you are doing are now in the smaller field, which is $GF(2^4)$. And that is much more efficient, well then when you are doing what an operation in $GF(2^8)$ ok. So, so therefore right there are also there are some constant multiplications. So, for example, α^{11} is a constant and so on. So, therefore these multiplications for these multiplications you do not need a dedicated multiplier, but you can actually do it much in a much more efficient manner ok.

So, so finally right you can also note that T^3 , when you are doing T^3 into so that is how we you are doing T^2 . But, when you are calculating T^3 right with multiplying it, you need not again multiply with T^3 , but rather you do at T^2 into this plus the original value like with what we did for the original mixed column operation instead of multiplying 3 into x we did 2 into x and then added x ok. So, you can again apply the same approach even in the composite fields ok.

(Refer Slide Time: 14:57)



So, so once you have done that essentially you have transformed the entire operations like you have transformed the mix columns, you have transformed the shift rows essentially will be simple, because it is just wiring, so there is nothing to be altered here. The bytes up or the sub bytes, we have already seen how to transform. The add round key again will be again similar, so you need not make any drastic any difference there. So, therefore right this is how your encryption flow will probably look like ok.

So, now I want to create its decryption algorithm. So, I suppose I want to implement in one single chip, I want to implement both the encryptor and the decryptor. Do you remember that this is a very you know like viable requirement, when you want to or you know like put your encryptor or encryption in a transmitter, which can work both of the transmission unit as well a receiver unit. So, you need both the encryption you need and the decryption you need in C 2 inside a single chip ok.

So, therefore right one thing you can observe here that when you are seeing the decryption operation, then this is just the decryption is just the reverse of encryption right. So, therefore here the final steps are observed if you observe right, you have an add round key. And then you are giving the cipher text, you are exporting the cipher text. This should be the first step, when you are doing decryption ok.

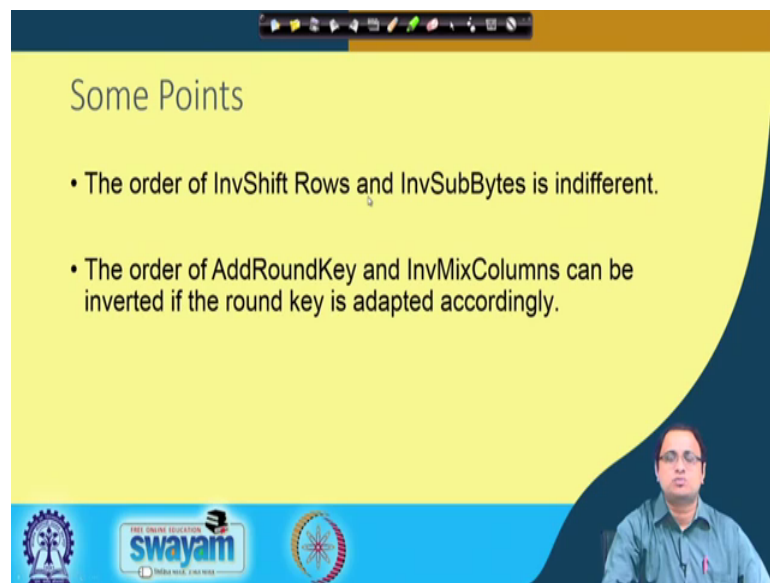
So, likewise note that in the first round, there is no mix columns or the I mean in the final round there is no mix columns. So, you have got a byte sub, shift row followed by add round key. When you are doing the inverse you are doing an inverse shift row, which just cancels out this shift row operation; you are doing an inverse byte sub, which cancels out this byte sub operation. And then you are again doing and doing an add round key or which is again that round key the inverse of the add round key is add round key itself, because add ground key is an idempotent operation. So, it is self invertible ok. So, you need not do any change over there, and then you essentially do the remaining steps ok.

But, we can easily observe that here if you have got a single chip to work as both, then the operations are not exactly synchronized. Like when you are doing a byte sub here, you are doing say an inverse shift row here. So, basically it implies that it looks like you need two separate dedicated paths for doing this to do this design in the in a single chip ok. And that may have an cost, because that may have you know like an overhead that that essentially can come up.

But, it turns out that for AES you can actually employ a couple of interesting very simple tricks, and you can rewrite this in this way. So, you can write this such that exactly they are balanced. Like if you are doing a byte sub here, you are doing an inverse byte sub here, if you are doing a shift row here, you are doing an inverse shift row here, if you are doing a mix columns here, you are doing an inverse mix columns here ok.

There is a small change that you have to do that the change is that for the round keys like which are intermediate that is if you leave out the first round and the you know the last round key, then you have to apply an inverse mix column operation to those. So, you have to slightly adjust the key scheduling ok, and that should be fine I mean it should be essentially able to get a decryption flow, which nicely synchronizes with the encryption flow. So how does it work.

(Refer Slide Time: 18:07)



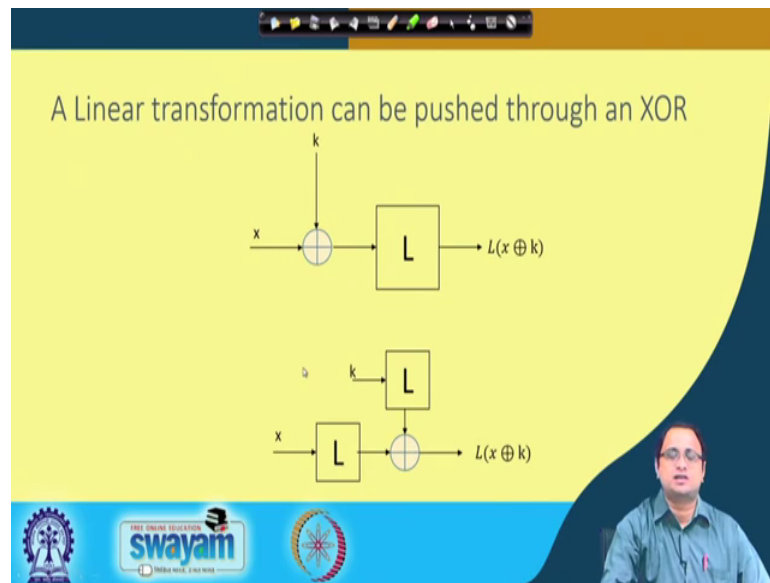
Some Points

- The order of InvShift Rows and InvSubBytes is indifferent.
- The order of AddRoundKey and InvMixColumns can be inverted if the round key is adapted accordingly.

swamyam

So, there are two important two simple points to observe here. The first thing is that the order of the inverse shift row, and the inverse shift bytes are indifferent ok. So, you can actually easily swap them ok. The second thing is that, the order of the add round key and the inverse mix columns can be inverted if the round key is slightly adapted ok. So, let me quickly talk about this point here, which may be little non-trivial to see.

(Refer Slide Time: 18:35)



So, I will try to explain this by this illustration. So, suppose you have got a simple linear transformation, which you can think of your mix column operation or mix column means like it is a linear transformation ok. So, therefore, I denote that by a matrix L , so that that the previous operation right say the add round key, which you have done prior to that is they denoted here as XOR. So, when you are doing x XOR with k , and then you are applying this L mapping to get $L(x \oplus k)$.

So, now suppose I want to in I want to swap these positions that means, I want to kind of push this XOR through the linear layer. If I do that, then essentially it turns out that I am basically bringing the L before and I am putting the pushing the XOR later on. But, it is easy to see that if I just make a small modification to my to the key that means, I passed the k also through a linear layer, then the output remains the same right, it is again L of x XOR k . So, therefore you can actually do this swap, but you have to just make a small change to this L layer ok. So, therefore depending upon whether you want to push the L before or you want to push the L later right, you essentially have to make a small adjustment to the secret key or to the round key so that is exactly being done here.

(Refer Slide Time: 19:51)



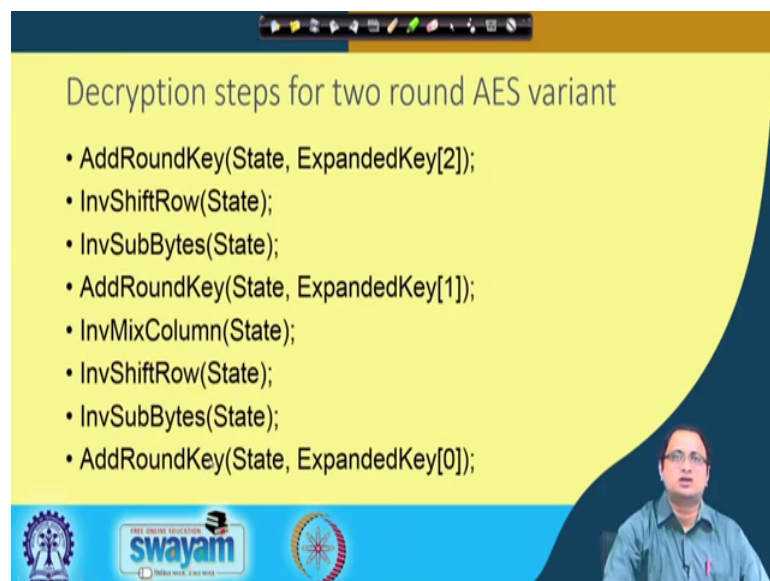
Encryption steps for two round AES variant

- AddRoundKey(State, ExpandedKey[0]);
- SubBytes(State);
- ShiftRow(State);
- MixColumn(State);
- AddRoundKey(State, ExpandedKey[1]);
- SubBytes(State);
- ShiftRow(State);
- AddRoundKey(State, ExpandedKey[2]);

The slide features a yellow background with a dark blue wave on the right side. At the bottom, there are logos for 'swayam' and other educational institutions, along with a small video inset of a man in a blue shirt.

And I try to explain this by a two round AES variant. Suppose, you take the add round key, where this is your input you have got the state, these are expanded key 0. Then you do a sub byte, then you do a shift row, and then you do a mix columns, and then again you do an add round key not again that in the last round you have only a sub byte and a shift row, and then you do an add round key ok. So, these are two round simple variant of the original AES ok. You can easily without loss of generality extend this to 10 rounds of AES 128.

(Refer Slide Time: 20:31)



Decryption steps for two round AES variant

- AddRoundKey(State, ExpandedKey[2]);
- InvShiftRow(State);
- InvSubBytes(State);
- AddRoundKey(State, ExpandedKey[1]);
- InvMixColumn(State);
- InvShiftRow(State);
- InvSubBytes(State);
- AddRoundKey(State, ExpandedKey[0]);

The slide features a yellow background with a dark blue wave on the right side. At the bottom, there are logos for 'swayam' and other educational institutions, along with a small video inset of a man in a blue shirt.

Now, if you consider the decryption unit of this, I have two again as I said work behind like this ok. So, I have to start with these do the inverse operation of these and so on. So, then it turns out exactly like these. So, I do the add round key now with the state expanded key² that is the final round key I do an XOR. I do an inverse shift byte, I do an inverse sub byte, I do an inverse shift row, then I do an inverse sub bytes, then I do an add round key, inverse mix columns, inverse shift row, inverse sub bytes, and finally an add round key.

Now, note that by the observation that we discussed just now or just prior to this you can actually change few of these sequences ok. For example, I can easily interchange the shift row and the sub byte operation. I can bring the sub bytes before, I can push the shift row operation down ok. And likewise I can also swap this inverse mix column, and the add round key. But, I have to make a small change in the expanded key ok.

So, when I am pushing the inverse mix columns right, before then I have to make I have to apply in order to cancel that effect I have to apply and I have to apply this operation also on the expanded key in order to counter that. So, whatever what I do is this, so therefore when I the moment I change this right see I have brought up the inverse sub byte pushed that the inverse shift row, I have changed these operations it is fine, but I only have to replace these expanded key by an equivalent expanded key, which is nothing but applying the inverse mix column on the key ok.

So, if you do this right, then essentially now and are now observed on this the way of writing in the decryption, you can see that it is exactly the same or it looks exactly like this the same as your decryption flow as your encryption flow ok. So, therefore, they are nicely balanced and therefore you can essentially kind of shared the hardware's and you can do them in parallel ok. So, once you are do you have understood this right, now we are in a position where we can share between the sub byte and the inverse sub byte, we can share between the mix column and the inverse mix column. And try to see whether we can save some gates ok.

(Refer Slide Time: 22:29)

Equivalent Decryption

- The equivalent key-scheduling can be obtained by applying InvMixColumns after the key-scheduling algorithm.
- This can be generalized to the full round AES.
- Thus we see that in the equivalent decryption the sequence of steps is similar.
 - This helps implementation

The slide features a yellow background with a dark blue curved border on the right. At the bottom, there are logos for 'swayam' and other educational institutions, along with a small video inset of a presenter in a blue shirt.

So, therefore right there are couple of interesting techniques, which people are proposed. For example, right so this is what we have already discussed that the equivalent key-schedule can be obtained by the apply the inverse mix columns, after the key-scheduling algorithm. And this can be generalized easily to the full round of AES. So, thus we see you know like that the equivalent decryption the sequence of the steps is similar. And this helps in implementing an encryptor and the decryptor on a single chip ok. You can essentially shared between the encryption and the decryption flow, and you can get a nicely optimized design.

(Refer Slide Time: 23:01)

Compact Encryption-Decryption Architecture

- Merge the operations in the encryption and decryption into a single unit.

The diagram shows a data flow: 'Input Data' enters a 'Data Schedule' block. An arrow from 'Key-Gen' points to 'Key Memory', which then points to the 'Encrypt/Decrypt' block. The 'Data Schedule' block also points to the 'Encrypt/Decrypt' block. The 'Encrypt/Decrypt' block points to the 'Dispatch Unit'. A feedback arrow labeled 'Enc/ Dec' goes from the 'Encrypt/Decrypt' block back to the 'Data Schedule' block.

The slide features a yellow background with a dark blue curved border on the right. At the bottom, there are logos for 'swayam' and other educational institutions, along with a small video inset of a presenter in a blue shirt.

So, so therefore right you can actually go for a compact encryption and decryption architecture. So, where you can merge the operations in the encryption and the decryption into a single unit. For example, right you can observe here that what I have tried to do is that I have tried to kind of represent this, if you compared with this with the architecture that I had in the last class, where I had a separate encryption and the decryption block. But, now we are trying to think that whether we can combine these two blocks into a single block ok. And that can save some gates, which is very important for our hardware designs, the rest of the things remain same ok.

(Refer Slide Time: 23:35)



Merged Round Operations

- *Addroundkey* is the same in both encryption and decryption
- *ShiftRows* and *InverseShiftRows* only differ in the direction of shifting

THE INDIAN EDUCATION swayam

INDIA 2020

So, now therefore let us try to see how we can merge. Add round keys of course the same in both encryption and decryption. The shift row and the inverse shift row only differ in the direction of shifting. So, they are pretty easy to kind of do it in a single hardware, you can just slightly program the hardware, and it should work in both the operations both in the encryptor unit as well as in is in the decryption unit.

(Refer Slide Time: 23:59)

Merged Mix Columns

A. Satoh et al., A compact Rijndael Hardware Architecture with S-box Optimization, ASIACRYPT 2001

$$\begin{pmatrix} c_3 \\ c_2 \\ c_1 \\ c_0 \end{pmatrix} = \begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix} \cdot \begin{pmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix}$$

$$= \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \cdot \begin{pmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix}$$

$$+ \begin{pmatrix} 08 & 08 & 08 & 08 \\ 08 & 08 & 08 & 08 \\ 08 & 08 & 08 & 08 \\ 08 & 08 & 08 & 08 \end{pmatrix} \cdot \begin{pmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix} + \begin{pmatrix} 04 & 00 & 04 & 00 \\ 00 & 04 & 00 & 04 \\ 04 & 00 & 04 & 00 \\ 00 & 04 & 00 & 04 \end{pmatrix} \cdot \begin{pmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix}$$

	Separate Mix (Inv) Columns	Merged Mix Columns
No. of XOR gates	152 + 440 = 592	195
Delay (gates)	5	7

But, what is interesting is to see how we can merge the mixed columns ok. So, there are couple of interesting approaches, which people are proposed. For example, right I mean this is an example where you are essentially trying to merge the mix columns ok. So, note that in inverse mix columns right the matrix the matrix is slightly different from the mix column matrix, it is like not slightly is actually significantly different from the mix column matrix. While in the mix column matrix you are nice operations like 2, 3, 1 and so on, which were easy to implement.

But, in the inverse mix column, we have got pretty non-trivial values which essentially are you know like are not easy to implement ok. For example, we have got the values like E B D a 9 and so on, which are not easy to implement. So, how do you essentially you know like can or how you can efficiently implement them, in particular when you already have a mixed column implemented ok. So, one of the approaches which is proposed by Satoh in Asia crypt in 2001 while he just follows, you write this matrix of inverse mix column.

And expressed it as summation of the original mix column matrix with two more factors. So, one of the factors is as shown here, and the second matrix is shown here ok. So, you can easily obtain the observe that these matrices are pretty easy I mean in the sense like they are all powers of two ok. So, therefore they should be easy to implement.

So, therefore right what you can do is that you can essentially have in one flow the but the normal matrix ok, you can have and in this other place you have this element this particular matrix. And therefore, right here you can get the original mix columns. But, when you bring in this particular thing and you XOR this with this, then you get the final inverse mix columns ok.

So, he basically kind essentially shared the or you know like utilize the origin or the already hardware, which is present in the mix columns rather than having two stand around you know one mix columns and one may inverse mix columns. So, when you are doing inverse mix columns, you are also using the hardware which is already there in your mix column matrix or for your mix column matrix.

So, it turns out that when you are doing a separate implementation, then it required something like 592 gates. Whereas, if you do this merging, then you need something like 195 gates ok, so that is a significant improvement. There is a slight increase in the delay, because now you have to wait there are the critical paths will slightly increase, but the critical path increase is not as much as the saving that you get, for example it goes from something like 5 gate units to 7 gate units ok.

(Refer Slide Time: 26:27)

Another InvMixColumn Design

$$\begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} 05 & 00 & 04 & 00 \\ 00 & 05 & 00 & 04 \\ 04 & 00 & 05 & 00 \\ 00 & 04 & 00 & 05 \end{pmatrix}$$

Gate-count around 166 XORs and a 32-bit MUX.

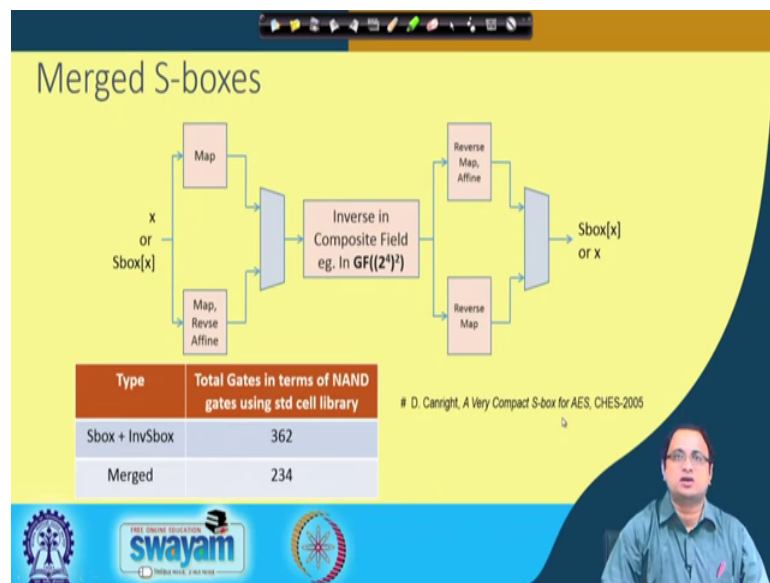
Subhadeep Banik, Andrey Bogdanov, Francesco Regazzoni: Atomic-AES: A Compact Implementation of the AES Encryption/Decryption Core. INDOCRYPT 2016: 173-190

There is another interesting approach, which was proposed later on which is essentially where you basically you know like factor this mix column matrix. So, if you factor this mix column matrix, then also you have got one of the factors as the mix column matrix,

whereas the other one is as follows shown here ok. And this is again you know like not very costly to implement. So, it turns out that this requires something like 166 XOR's, and of course there is 32 bit MUX ok. So, here also you know you have around 195 units, and here you require around 166 XOR's ok.

So, therefore you basically have got a MUX which is over there, and then you follow it by an AES mix column operation. So, in one case you are basically passing you through this matrix followed by a mix column to give you this inverse mix column operation. On the other case when you are just using it for the encryption part that means when you want it to configure get configured as only the mix column, then you have to bypass this operation. So, you basically go through this path, and you just apply the mix columns ok. So, again you have got a shared mix column and inverse mix columns operation.

(Refer Slide Time: 27:35)



So, likewise right you can also merge the S-boxes. So, when you merge the S-boxes, then again there are certain you know like certain improvements that you can do. But, so likewise right as we have seen that how we can merge the mix column operation, and the inverse mix column operation. So, likewise we can also merge the S-box and the inverse S-box operations ok, and that can also lead to opportunities of saving.

So, here is an example how we can do that. So, if you remember like when you are doing the X or the S S box X. So, when you are doing the forward transformation, so you take X, you map this X say to GF of 2 power of 4 square, which is in the composite field.

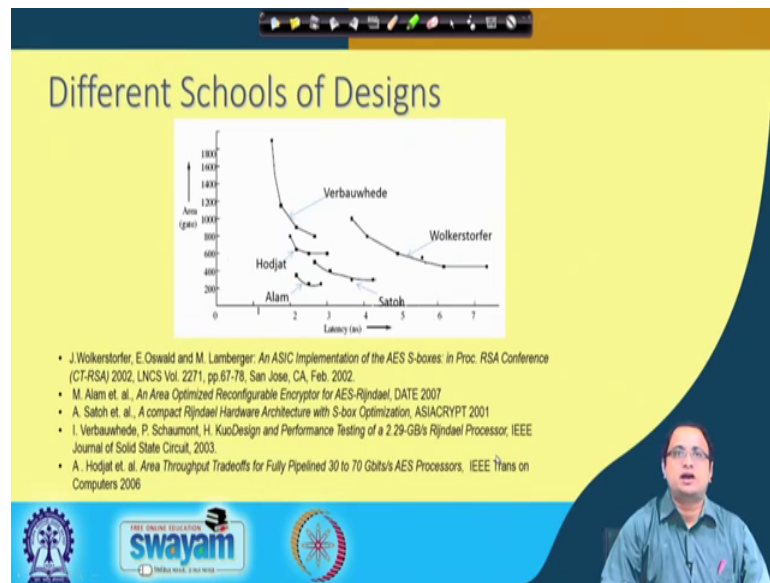
Suppose you do the inverse in the composite field ok, and then you do the reverse map ok. So, then you bring and so if you do that right, then you get X, I mean you get S-box X.

So, likewise right when you are doing decryption, you take S box X, you do the inverse operation that means, the inverse mapping. You apply the reverse affine operations, you can combine these two things to save some gates, because both of them are matrices. And then you pass them through an inverse, which is in the composite field $GF(2^4)$.

And finally, you do the reverse mapping to get the result back, which is in this case X. So, you have basically fail in S-box X, and it since you are doing the inverse operation you are getting X back so it is fine, because the core operation in both the S-boxes as well as the inverse S-box is the finite field inverse ok. And that is the major hardware which you are trying to share in both the data parts ok.

So, for example like if you go now for you if you compare it with say one standalone S-box and an inverse S-box, then you would probably require something like 362 NAND gate equivalents in a standard cell estimation. If you combine them and merge them, then it reduces to something like 2034, so that is a significant amount of reduction in the gate count ok. So, so therefore right I mean of course, you need to implement the inverse in an efficient manner, and that is essentially studied in several works. In particular it is nicely described in this paper by Canright, which is a very compact S-box for AES and it is a very pioneering work and how you can efficiently implement the S box in composite fields ok.

(Refer Slide Time: 29:51)



So, so therefore right I mean the there are different schools of design, and here is a quick snapshot on the objectives. So, therefore some people have worked for latency, some people have worked for area, so therefore you know like you would probably try to place your design into different points in this spectra depending upon your application ok.

So, here are some bunch of interesting references, which people have you know like developed to design efficient AES implementations ok. So, therefore for example, right I may my focus may be only to make it very high speed in which case I should be looking for designs, which are in this part of the spectrum. Whereas, if I want a lightweight design, then I should be looking maybe at this point of the spectrum. Whereas, if I want both, then probably I should be to gain these points of the spectrum, which are which kind of nicely trades off both area as well as latency ok.

So, so in particular right this implementation which is shown here by Alam in date 2007 right. Then here in this design we try to develop an AES, which supports both one I mean for all the three configurations of AES, like 128, 192, and 256, and still essentially trades off nicely the area as well as latency ok. So, it can be an interesting read for our purpose.

(Refer Slide Time: 31:07)

State of the Art

- Intel has introduced dedicated hardware for AES on its microprocessors
- These are accessed by dedicated AES instructions
- The AES design consists of:
 - Standard techniques used such as composite fields, single map – reverse map, etc.
 - Speed of 53Gbps obtained by fully custom design flow on 45nm technology.

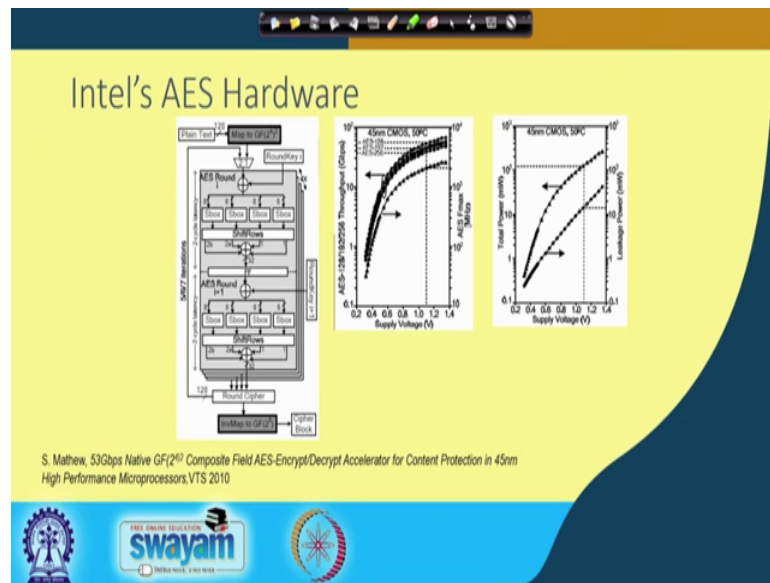
S. Mathew, 53Gbps Native GF(2⁸) Composite Field AES-Encrypt/Decrypt Accelerator for Content Protection in 45nm High Performance Microprocessors, VTS 2010

swamyam

So, so let us mean let me just conclude with the state of the art discussion. So, Intel has introduced dedicated hardware for AES on its microprocessors. These are accessed by dedicated AES instructions often called as AES NI. So, this AES designed consists of standard cell design approaches standard cell techniques.

So, it is a very interesting a very nice piece of engineering work. The techniques that are followed again are essentially what we have already been studying in this class like composite fields, single map, reverse map. So, these are is exactly that what we have studied. But, the engineering is so nice that it ends up in realizing a speed of 53 Gbps obtained by a fully custom design flow on 45 nanometer technology. So, this was published in VTS 2010 so some of you were interested can go through these paper.

(Refer Slide Time: 31:59)



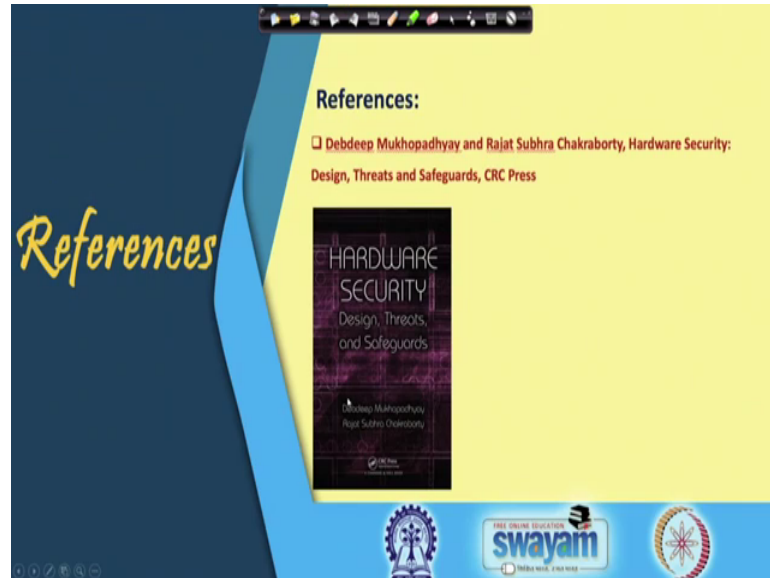
Here is a you know sort of a pictographic diagram about how the architecture looks like. So, if you see now carefully, the ideas are essentially what we have already being studied. For example, the plain text is mapped to GF 2 power of 4 squared, and then the operations are done in the composite in composite fields ok. So, there are all these S-boxes which are placed over there.

And this is an and finally right, there are I mean depending upon the configurations there are different iterations, which are set like there are either 5 iterations or 6 iterations or 7 iterations. And finally, you get back the result which is in the composite field. So, you do an final mapping or the reverse mapping from GF 2 power of 4 square to GF 2 power of 8. So, therefore this mapping right like which is shown here by the shaded boxes are done only once. So, you are not doing any intermediate transformation. So, you are representing the entire AES in composite field. And the you know as we have discussed like the mix columns, the affine transformations for sub bytes all of them are transformed into composite fields ok. So, the entire operation is done in composite field.

And that essentially gives an opportunity of a very high throughput AES implementation ok, which can be realized in your inter in our Intel processes. And as we shall discuss later on right this essentially also gives us lot of nice observe nice properties in the sense like it not only reduces latency, but it also helps to be you to protect against several side

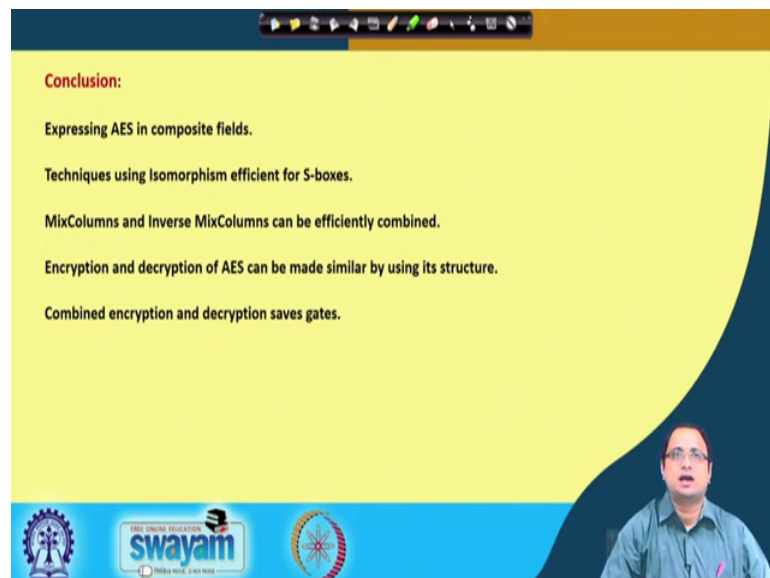
channels, in particular software side channels, like cache diving attacks which we will be discussing in our future classes.

(Refer Slide Time: 33:29)



So, let me stop here. So, again this is my standard reference that I have followed for the book, I mean follow for the car for the follow for the class.

(Refer Slide Time: 33:37)



And to conclude like we have been discussing about how we can express the entire AES in composite fields. We have discussed about techniques using isomorphisms, which are efficient for S-boxes. We have discussed about how we can combine officially the mix

column and the inverse mix column operations. We have discussed about how encryption and decryption of AES can be made similar by using its structures. And finally, we have discussed about how I mean we I mean the general observation is that when you are combining encryption and decryption, then it says valuable gates ok. So, with this I would like to stop here.

And thank you for your attention.